(2)

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

PARTICLE-SIZING SYSTEM FOR
SCANNING ELECTRON MICROSCOPE IMAGES OF
SOLID-PROPELLANT COMBUSTION EXHAUST

by

Yeaw-Lip Lee

March 1991

Thesis Advisor: John P. Powers

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No / Project No / Task No / Work Unit Accession Number

11. TITLE (*Include Security Classification*)
PARTICLE-SIZING SYSTEM FOR SCANNING ELECTRON MICROSCOPE IMAGES OF SOLID-PROPELLANT COMBUSTION EXHAUST (U)

12. PERSONAL AUTHOR(S) LEE, YEAW-LIP

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	March 1991	149

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy of the Department of Defense or the U.S. Government

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	Particle sizing, SEM, image digitization, image segmentation, rocket motor, C language

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

Accurate measurement of particle size distribution of rocket motor exhausts is essential for predicting the combustion efficiency and infrared plume signature. This thesis presents an automated method for extracting particle size distribution from scanning electron microscope (SEM) images. The SEM images were taken off a filter paper placed at the end of a collection probe inserted into the exhaust plume. The automated SEM extraction system consists of an IBM AT-based computer system fitted with a 512 x 480 pixel frame grabber. Photographic images taken off the SEM are acquired via a vidicon camera. A C language program was written to control the hardware and automate the extraction process. A threshold is first applied to the digitized image and the resulting binary image is subjected to object segmentation. Each object is then sized and the distibution from one or more images can be plotted. The main bulk of this thesis is to document the software specially written to undertake this set of tasks. Results obtained were compared with that from a Malvern

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL John P. Powers	22b. TELEPHONE (*Include Area code*) (408)-646-2679 / 22c. OFFICE SYMBOL EC/Po

MasterSizer particle sizer and found to be favorable. Particles as small as 1/8 microns have been successfully sized.

Approved for public release; distribution is unlimited.

Particle-Sizing System for
Scanning Electron Microscope Images
of Solid-Propellant Combustion Exhaust

by

Yeaw-Lip Lee
Civilian, Singapore Ministry of Defense
B.Eng. (Elect), National University of Singapore, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

Author: _____
Yeaw-Lip Lee

Approved by: _____
John P. Powers, Thesis Advisor

_____
David W. Netzer, Second Reader

_____
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

Accurate measurement of particle size distribution of rocket motor exhausts is essential for predicting the combustion efficiency and infrared plume signature. This thesis presents an automated method for extracting particle size distribution from scanning electron microscope (SEM) images. The SEM images were taken off a filter paper placed at the end of a collection probe inserted into the exhaust plume. The automated SEM extraction system consists of an IBM AT-based computer system fitted with a 512 x 480 pixel frame grabber. Photographic images taken off the SEM are acquired via a vidicon camera. A C language program was written to control the hardware and automate the extraction process. A threshold is first applied to the digitized image and the resulting binary image is subjected to object segmentation. Each object is then sized and the distribution from one or more images can be plotted. The main bulk of this thesis is to document the software specially written to undertake this set of tasks. Results obtained were compared with that from a Malvern MasterSizer particle sizer and found to be favorable. Particles as small as 1/8 μm have been successfully sized.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Inasmuch as many have undertaken to compile an account of the things accomplished among us, just as those who from the beginning were eyewitnesses and servants of the word have handed them down to us, it seemed fitting for me as well, having investigated everything carefully from the beginning, to write it out for you in consecutive order, most excellent Theophilus; so that you might know the exact truth about the things you have been taught. (The Gospel According to Luke, The Holy Bible)

The above passage has provided me much inspiration and motivation to write this thesis. I would like to thank my thesis advisor, Prof. John Powers, and my second reader, Prof. David Netzer, for sharing their knowledge and for their careful guidance. CAPT Lyle Kellman has been especially helpful in explaining the intricacies of collecting exhaust samples. Many thanks go to him for the many painstaking hours spent in front of the SEM to provide me the SEM photographic images used in my work. I would also like to thank Kim Tran for her assistance in reproducing some of the figures. Finally, I would like to thank my wife, Chew-Leng, for her unwavering support and for her assistance in proof-reading this thesis.

I. INTRODUCTION

The rationale for analyzing scanning electron microscope (SEM) images of combustion exhaust stems from the on-going research to investigate particle behavior in exhaust nozzles and plumes of solid propellant rocket motors. Netzer and Powers describe various methods being tried in the quest to obtain accurate particle size distribution [Ref. 1]. Accuracy is essential because the various computer codes available for predicting performance and infrared signatures are highly sensitive to the particle size distribution.

Traditionally, SEM images are analyzed visually by a trained observer, with the aid of linear scales (stage or eyepiece micrometer), graticules (British Standard BS 3406), or other visual cues. Full use is made of the human observer's ability and talent to differentiate shapes and features. Humphries [Ref. 2] has a concise description of the various methods. Disadvantages include low productivity, fatigue, observer subjectivity, and a high probability of error. The generally small sample population which a human observer is able to analyze also results in sampling errors due to poor statistical averaging.

The standard procedure in particle size analysis (PSA) is to allot each particle to its appropriate size class and, from the resultant data, calculate (or determine graphically) parameters that adequately describe the size distribution. These parameters include particle projection lengths, perimeter, form factor, area and volume. Automatic PSA offers a significant advantage over manual microscopy in

that it allows large numbers of particles to be measured consistently and accurately. However, automatic PSA is not without limitations. Resolution and contrast performance of the imager is generally poorer than the human eye. Also, the instruments for such systems tend to be rather elaborate and expensive.

The current effort is a spin-off from work done with holograms, first by Redman [Ref. 3] and Orguc [Ref. 4] using the FORTRAN language, and subsequently by Kaeser [Ref. 5] and Hockgraver [Ref. 6] in the C language. The C algorithms used in this thesis have been optimized for speed, yet have reduced false feature identification. In the earlier work with holograms, significant problems were encountered with speckle in the reconstructed hologram images. This limited the particle size determination to particles larger than 10 microns [Ref. 7]. The current work attempts to analyze particles down to the resolution limits of the imaging system. For the SEM images analyzed, particle size distributions down to one-eighth of a micron have been achieved.

This thesis is divided into six chapters. Chapter II describes the SEM images, what they represent and how they are obtained. Chapter III provides an overview of all the hardware and software required to support this work. Chapter IV goes into the detailed description of the program modules making up the Scanning Electron Microscope Extraction (SEMEX) program which forms the bulk of the current effort. Chapter V describes the procedures required to acquire a SEM image from a photograph, to process it to a form suitable for extraction of particles, and, finally, to extract and compile the results for plotting or further statistical analysis. Chapter

VI describes the experimental results beginning with calibration, speed performance comparison with earlier programs, and finally summarizes the results from SEM images obtained from an actual motor burn. The latter are correlated with results obtained from the Malvern MasterSizer particle sizer [Ref. 8]. The last chapter lists the conclusions arrived at and recommendations for future efforts in this area.

II. SEM IMAGES OF COMBUSTION EXHAUST

This chapter introduces the combustion mechanisms involved in metallized solid-fuels and solid-propellants, and the need for accurate measurement of the plume particle size distribution. One such method using SEM images of the collected exhaust is described.

A. COMBUSTION PHENOMENA

The higher energetic performance of metallized solid-fuels (i.e., fuels containing boron, magnesium, titanium, aluminum, etc.) has sparked considerable interest in solid-fuel ramjets. While the exact combustion phenomena for metallized fuels has not been fully understood, Gany and Netzer [Ref. 9] describes one possible scenario, shown in Figure 1. During combustion, the hydrocarbon fuel vaporizes at the surface exposing the metal fuel particles. As there is little or no oxygen at the

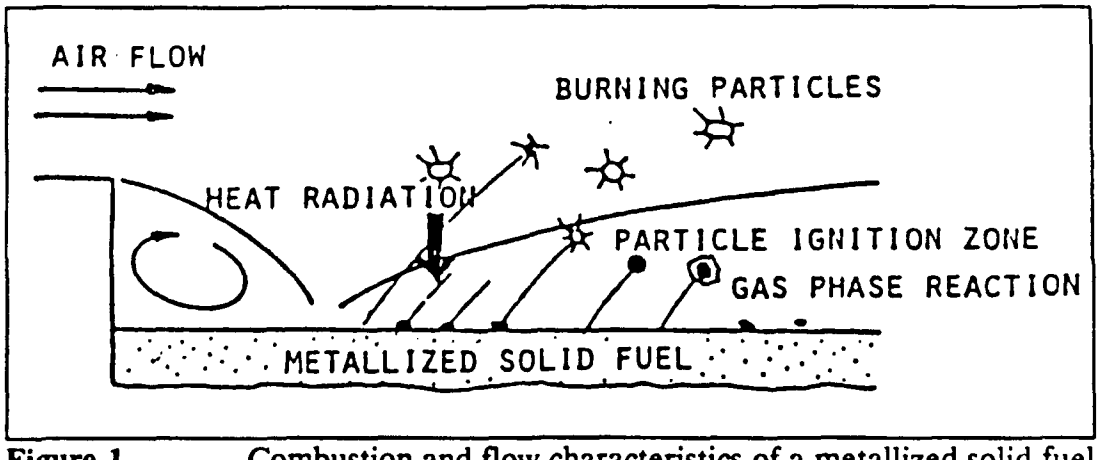

Figure 1. Combustion and flow characteristics of a metallized solid-fuel [From Ref. 9:p. 424].

fuel surface, these metal particles can heat up but will not ignite. As more of the surrounding hydrocarbon fuel vaporizes, the metal particles tend to coalesce before being ejected into the main flow which is rich in oxygen. A particle may collide with other particles and agglomerate, or it may contact oxygen and ignite intensely. Agglomeration of metal particles may be one reason for poor combustion efficiency. Another may be due to the oxide coating on the particle, thus slowing the rate of chemical reaction. Agglomeration and oxidation also result in a large variation in particle sizes.

Metals are also added to solid-propellants for rocket applications to provide combustion stability and/or increased performance. Aluminum is the most often used performance enhancer. The particles burn to produce aluminum oxide of varying sizes. When these particles pass through the exhaust nozzle, they can change in size distribution due to breakup and collisions. This two-phase flow results in a loss in delivered thrust. When these particles exit the nozzle into the plume, they significantly affect the plume radiation. The infrared signature of the exhaust plume is a function of the temperature profile of the plume and this, in turn, is dependent on its composition of hot gases and metal particles. From high-speed infrared imaging, it is noticed that the center of the plume appears hottest. Some of this is due to afterburning of the fuel-rich exhaust gases with the ambient oxygen. It is also suspected that the larger particles cannot turn as rapidly as the gas within the exhaust nozzle, resulting in these particles being located nearer to the plume centerline. They also have a greater heat capacity and, hence, cool off slowest after exiting from

the exhaust. Accurate prediction of plume radiation requires accurate knowledge of the particle size distribution at the nozzle exit. To validate the codes, axial and radial variations in the plume particle size distribution need to be accurately measured.

The need for accurate determination of particle size and their distribution is recognized and accurate methods are available to determine modal distributions. However, a complication lies in the multi-modal distributions caused by the wide range of particle sizes expected. Particle sizes may vary from sub-micron sizes to several tens of microns. For particles larger than about 2 μm, various methods, such as forward laser-diffraction measurements [Ref. 10] have been successful. For sizes down to 0.5 μm, the Malvern MasterSizer particle sizer has been used; utilizing Mie corrections to the diffraction equations [Ref. 8]. Measurements using these techniques have not been successful for particles in gas flows smaller than 0.5 μm because of the requirement for short focal length lenses and the resulting restricted measurement volumes. Analysis of the collected exhaust particles using the SEM avoids these problems and shows promise as it allows for very clean images even at high magnification. Resolutions down to one-eighth of a micron have been possible, as is shown in this work.

B. SOURCE OF SEM IMAGES

The SEM images are extracted from combustion exhaust taken from a small (5 cm) experimental rocket motor developed by the Naval Postgraduate School. Different propellant compositions are being tested. The firings and subsequent

extraction of the SEM images were carried out by CAPT Lyle Kellman, a thesis student working under Prof. David Netzer. A detailed description of his work is found in his thesis [Ref. 11]. Each firing produces two sets of Malvern MasterSizer data, one generated directly during the burn and another from dissolving the filter paper removed from the collection probe. Part of the filter paper from the collection probe is also analyzed under the SEM. The resulting SEM images are the inputs to SEMEX and the focus of this thesis study.

1. Experimental Setup

The combustion exhaust from the rocket motor is sampled with the aid of a collection probe. The probe is placed at a desired location behind the exhaust nozzle and is protected from the intense heat of the exhaust by a plume deflector. The deflector prevents the plume from impinging on the collection probe tip until the rocket motor achieves steady-state, and is then lowered for one second. The exhaust enters the collection probe through a small entry nozzle and expands inside, depositing the combustion products on a filter paper. At about the same time, the Malvern MasterSizer is activated to scan the captured exhaust plume and to store its results for later analysis. A schematic of the setup is shown in Figure 2.

2. Preparation of SEM Specimens

After firing, the filter paper is removed from the rear of the probe. Two half-inch diameter samples are cut out from the edge and center of the filter paper. These samples are gold-plated in preparation for scanning under the SEM.

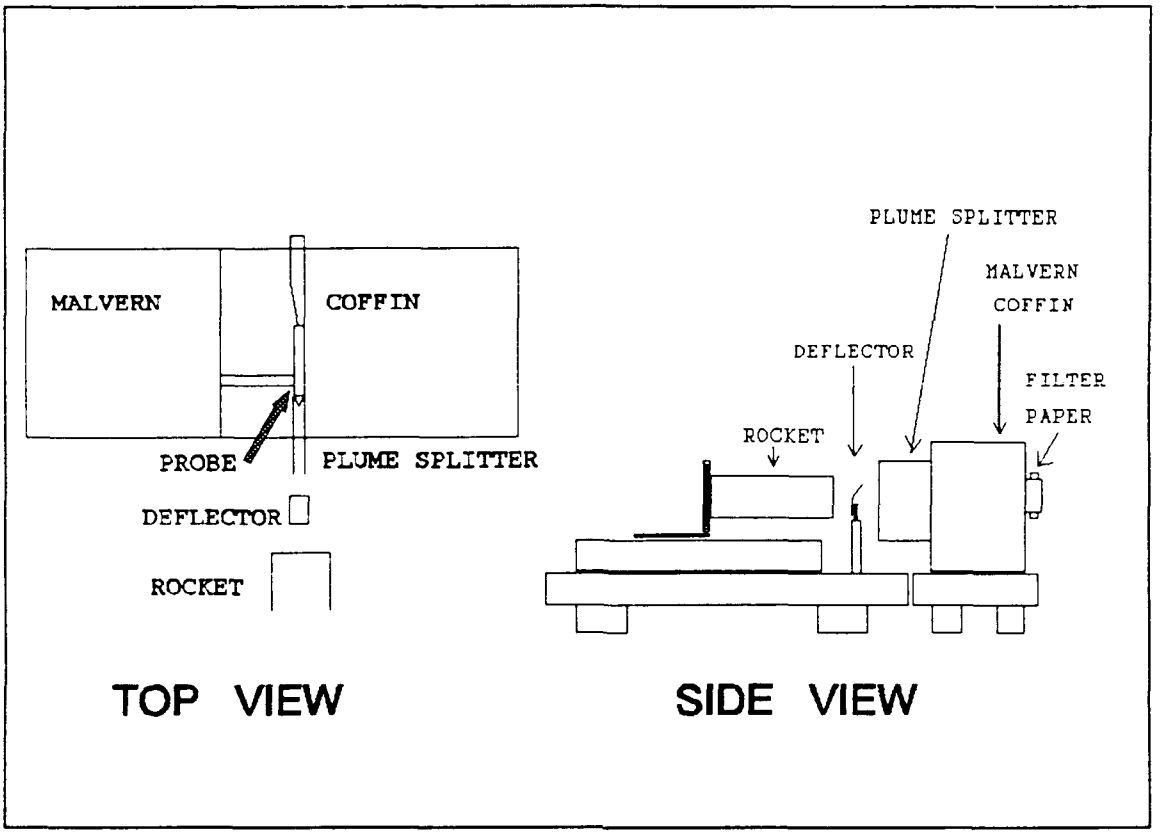

Figure 2. Experimental setup to collect exhaust particles from a solid-propellant rocket motor [Ref. 11].

The rest of the filter paper is dissolved in acetone. The metal particles are allowed to settle to the bottom and the supernatant liquid is siphoned off. This process is repeated several times until the paper is fully dissolved and removed. In the final step, acetone is added again and the mixture of acetone and solid particles is agitated to get a homogeneous mixture. A drop of this mixture is extracted and analyzed with the Malvern MasterSizer. This produces a second set of Malvern results.

One source of error comes from dirt or other debris being introduced on the filter paper during combustion and subsequent handling. Sampling errors can

occur if the half-inch samples are not taken from representative portions of the filter paper where particles are uniformly distributed.

3. Photographing the SEM Images

The prepared specimens are placed inside a Hitachi Model S450 SEM and the SEM chamber evacuated to a high vacuum. The specimens can be moved about in search of representative portions of the specimens. This is carried out with the SEM on low magnification. The area being scanned is displayed on a high-persistence CRT. Once particles are found, the magnification is increased to 1,000. The scanning electron micrographs are then recorded on Polaroid photographic film for subsequent analysis by SEMEX. The resulting images have between 5 and 1,000 particles, depending on the type of propellant used, the time of exposure, the distance of the collection probe from the exit nozzle, the radial position of the probe, and the location of the filter paper from which the sample was taken. To increase the number of particles on a single photograph, multiple exposures were taken from different fields. It was found that up to four exposures could be taken without significant degradation to the image. The SEM was set for maximum contrast and minimum brightness so that the background stays relatively dark even after four exposures.

A source of error comes about from the choice of the area to be photographed. This is a subjective task. Many of the areas may contain only a few particles while a few areas may contain a large splattering of particles. Each photographed area (70 μm by 80 μm) only represents about 1/22,500th of the sample

area and typically only 30 to 40 different areas can be photographed in a 4-hour session. Thus, the size distribution of the particles can be heavily biased by the SEM operator if the particles are not evenly distributed.

C. CURRENT EFFORTS

Current work carried out by the Naval Postgraduate School for the Air Force Phillips Laboratory include analysis of the exhaust plume by high-speed video and infrared cameras, programmed data acquisition of temperature and pressure using transducers, and analysis of the exhaust plume composition by the Malvern MasterSizer and SEMEX. The experimental data will be used both as inputs to and for validation of plume prediction computer codes. The work is primarily carried out by the Aeronautics Department with the Electrical and Computer Department assisting in the area of SEM image digitization and feature extraction.

In this thesis, the work done encompasses the development of the SEMEX program and the setting up of a reliable methodology for the digitization and feature extraction of the SEM images. Calibration of the system has been carried out and comparisons have been made with the Malvern MasterSizer. The performance of several SEMEX modules (in particular, *CLIP*, *TAG* and *SIZE*) have also been compared with relevant portions of the HOLOGRAM C program developed by Kaeser [Ref. 5] and improved upon by Hockgraver [Ref. 6]. The results have been favorable.

III. OVERVIEW OF THE AUTOMATED SIZING SYSTEM

The automated particle sizing system consists of an IBM AT microcomputer fitted with dedicated hardware and run by a program written in the C language. Figure 3 shows the equipment involved in the extraction of particle size data from

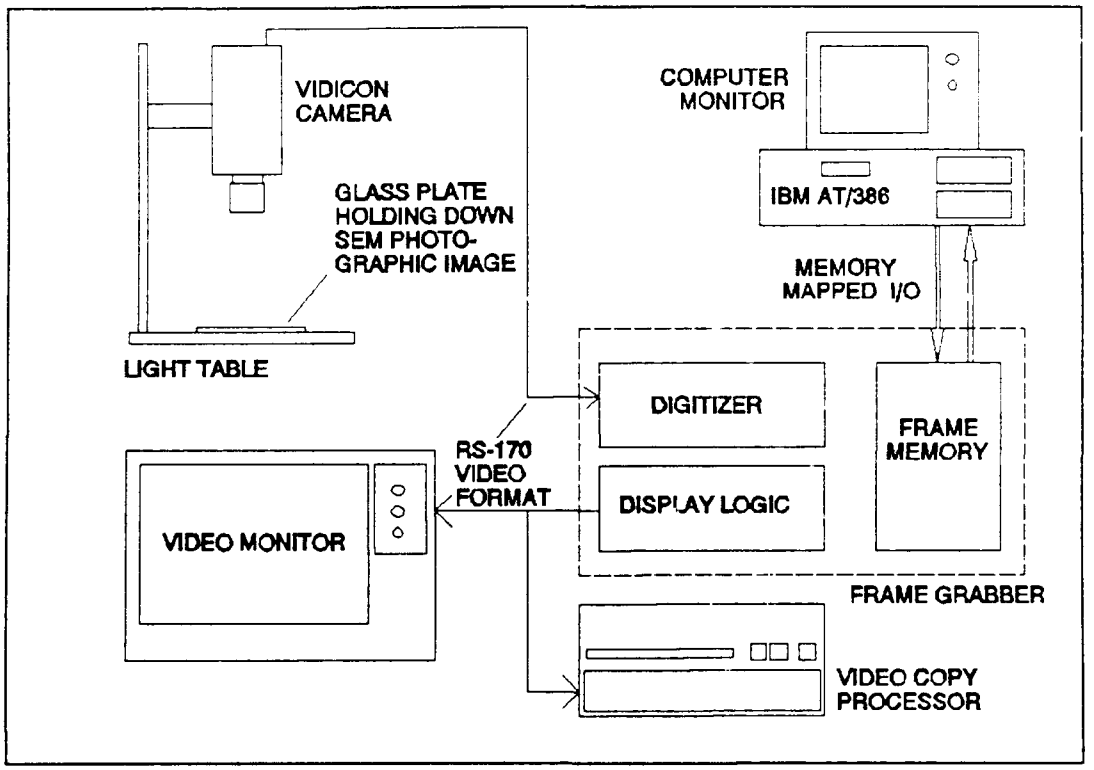

Figure 3. Diagram showing the equipment setup used for extracting particulate size data from SEM photographic images [After Ref. 12].

SEM photographic images. SEM photographic images are acquired by the vidicon camera and digitized by the frame grabber. Processing of the images is carried out by the IBM AT and the processed image is displayed on the video monitor. Digitized images in the frame memory can be saved to disk for later retrieval. User

dialog is carried out through the use of the computer monitor and the keyboard. A hardcopy of the digitized image can be printed on the video copy processor. The details of the hardware configuration and the software support are described in the rest of this chapter.

A. HARDWARE CONFIGURATION

The system configuration consists of the following hardware:

- IBM AT with Intel Inboard 386/AT accelerator board and EGA monitor. The Inboard 386/AT disables the Intel 80286 microprocessor inside the IBM AT and executes programs with the 32-bit Intel 80386 microprocessor running at 16 MHz. This enhancement gives a speedup of 17.6 over a standard IBM PC. The computer also has a 80287 math coprocessor running at 10 MHz to speed up floating point operations. The EGA monitor functions in text mode and displays the dialog boxes. The user makes his or her choices by typing in commands on the keyboard.

- Imaging Technology PCVISION*plus* frame grabber board. This is a video digitizer and frame memory capable of digitizing the standard RS-170 composite video input from the vidicon camera. Figure 4 shows a block diagram of the frame grabber. Two camera inputs are available. Each camera input, in turn, has three input channels (red, blue, and green) to handle pseudo-color images. However, only the green channel is being used for monochrome images because it carries the synchronization signals. The built-in gain and offset circuits can be adjusted through software control to achieve optimum brightness and contrast. The look-up tables (LUTs) are special memories used to transform pixel values without altering the contents of frame memory. Although there are 32 LUTs (8 input LUTs, and 8 output LUTs for each of the red, blue and green channels), only one input LUT and one output (green) LUT are required. The digitized image, stored in a special on-board high-speed memory (called the frame memory), can be manipulated by specially written software functions and saved to a disk file for later analysis. The frame memory can handle up to two images, each 512 x 512 pixels wide. However, the RS-170 video signal can only provide 512 pixels by 480 lines. Therefore, the 481st to 512th rows of the frame memory are not used. The digitized image has to be converted back to analog RS-170 video signals before the image can be displayed on a composite video monitor. [Ref. 12]

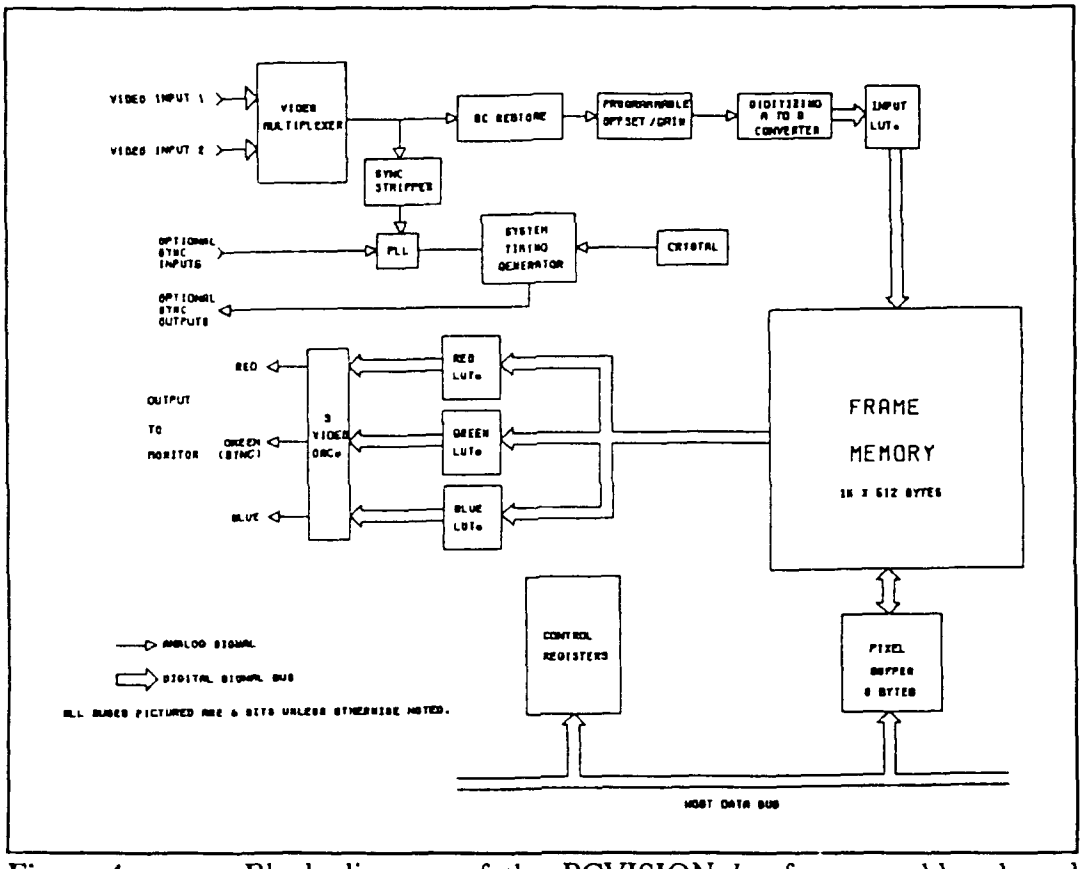

Figure 4. Block diagram of the PCVISION*plus* frame grabber board [From Ref. 12].

- Panasonic Model WV-1410 CCTV camera and light table. The Closed-Circuit Television (CCTV) camera uses a 525-line vidicon imaging element with 25 mm f/1.4 lens and is able to operate down to 5 lux incandescent illumination. The internal electronics scan and convert the image into composite monochrome video which is routed to the frame grabber. The light table (not shown) consists of an adjustable mount for the camera and four 75 W incandescent lamps fitted with diffusers. Setting the height at approximately 13 inches from the SEM photograph enables the 3.5" by 4.5" photographic image to be captured onto the 512 x 480 pixel video frame. The camera and illumination setup is critical for obtaining a good digitized image. A piece of flat transparent glass, used to hold the photograph flat, completes this setup. Dust particles should be excluded. The placement of the lamps is crucial for obtaining even illumination. A detailed procedure is provided in Chapter V.

- Panasonic TR-196M monochrome composite video monitor. This is used to display the digitized SEM image and any video operations carried out on it.

on it. The horizontal size of this particular monitor cannot be adjusted to display the full frame. Hence, the left and right edges are not visible and can only be seen using the video copy processor. This is a serious shortcoming as edge artifacts caused by poor light placement cannot be detected readily.

- Tektronix HC01 video copy processor. The video copy processor accepts composite video from the output channel of the frame grabber board and is used for making a hardcopy of the video image displayed on the composite video monitor. The video copy processor uses a thermal process to etch the video image onto 4-inch wide heat-sensitive paper.

- IOMEGA Bernoulli Box II dual removable cartridge disk (RCD) system. Each RCD consists of two flexible disks enclosed in a plastic housing providing 20 MB of formatted disk storage. As each digitized image takes between 100 kB to 250 kB in a compressed form, this allows storage of approximately 100 images in each RCD. One drive is dedicated to storing image and data files. The data files contain the tabulated particle sizes (*.dat* files), the histogram results (*.his* files) and the session activities (*.ses* files). The other drive contains the executable SEMEX file and the C source codes for all the modules, together with the library files and the software development system. A 44 MB version of the RCD is also available.

- HP LaserJet series II printer for producing a hardcopy of the results and the histogram plots (optional).

B. SOFTWARE SUPPORT

The software used to support this work include:-

- Microsoft C Optimizing Compiler, version 5.1, with Large Memory Model Run-Time Library, CodeView Symbolic Debugger and Overlay Linker. This provides the C programming language and the software development system for the SEMEX program. The compiler converts the C source files into object files which are linked together with the libraries by the linker. The large memory model library *(LLIBCE.LIB)* is recommended by Imaging Technology to support their ITEX PC*plus* software. This model allows for multiple 64 kB segments for both code and data to contain the large image and data arrays. CodeView aids in the debugging process by allowing single-step program operation while observing key variables. Microsoft also has a *MAKE* facility that automates the compilation and linking process through the use of a special batch file shown in Appendix A.

- Star Guidance Window BOSS windowing package. The Window BOSS is an extensive library of C functions for the creation, management and manipulation of text windows, which are essentially dialog boxes for program-user interaction. It allows for multiple windows to be created using dynamic memory allocation and manages these windows. Window attributes like border styles, border colors, and window foreground and background colors can be set. Context-sensitive help screens can be easily implemented. Forms and input functions are also available to aid the programmer in developing user-friendly dialog boxes. The Window BOSS is a shareware product. [Ref. 13]

- Imaging Technology ITEX PC*plus* Large Memory Model (*ITEXPCML.LIB*) Library. The ITEX PC*plus* is a library of image processing and graphics functions specially written to support the PCVISION*plus* Frame Grabber. Only a few of the functions are actually used in SEMEX and these include setting up of the frame grabber's control registers and LUTs, reading and writing a single pixel value, reading and writing the frame memory from and to disk, and thresholding an image. Details can be found in the ITEX PC*plus* Programmer's Manual [Ref. 14].

- Mathworks MATLAB 386. This mathematical programming package is used to produce the histogram plots subsequent to the *ANALYZE* stage. A script file was written to automatically read in a SEMEX data file and plot the results. It can also plot the Malvern data. This file appears at the end of Appendix B.

- SEMEX (Scanning Electron Microscope EXtraction) Program is a set of program modules, written in the C programming language as part of this thesis work. This program comprises thirty-seven locally-written functions and handles the six stages required in extracting particle size information. The first involves setting up the system to ensure correct illumination, camera input gain and offset, and calibration. The second stage deals with the digitization of SEM images, cropping the digitized image to the desired area of interest, and complementing the image, so as to obtain a digitized image of the area of interest with particles dark against a light background. The third stage employs thresholding to form a binary image. The fourth and fifth stages involve particle tagging and particle sizing respectively. Here, particles are identified from the background and their x-chord and y-chord lengths and areas are measured. The final stage merges the data from several images and analyzes the resulting data. The analysis involves extracting three-dimensional features from the two-dimensional images. Presently, this last stage has been written to generate results similar to the Malvern MasterSizer. The latter puts out a histogram plot of percentage of particle volume against a logarithm scale of the particle diameters. SEMEX generates the histogram data and makes use of MATLAB to generate the histogram plots. The source listings for the various

modules are found in Appendix B and detailed descriptions are given in Chapter IV.

IV. DETAILED DESCRIPTION OF SEMEX

SEMEX is the name given to the executable file made up of the seven C language program modules described in the following sections. SEMEX stands for Scanning Electron Microscope EXtraction. SEMEX was written with the aim of providing a user-friendly environment which is flexible, yet highly efficient for digitizing SEM photographic images and extracting particle size distribution data. Windows (dialog boxes) guide the user through the whole process of setup, acquisition, processing (more specifically clipping, tagging, and sizing), and analysis as shown in Figure 5. Generally, *SETUP* and *ANALYZE* need only be done once at the beginning and at the end of a session, respectively. *ACQUIRE, CLIP, TAG,* and *SIZE* need to be done as many times as there are images to extract.

Formatted inputs disallow illegal user responses while extensive error trapping prevents unintentional user inputs from resulting in fatal problems. A record of each session's activities is automatically created in an ASCII text file format which can be reviewed or printed out as a permanent record by any text editor program. The *sessions* file uses a filename formed from the current date (i.e., the first three letters of the month and a two-digit day) with the extension of *.ses*. In this way, each day's activities are recorded in a separate file. If more than one session is started in a single day, the second session will be appended to the end of the first. This prevents any record from being written over. Alternatively, the user can type in a filename during setup.

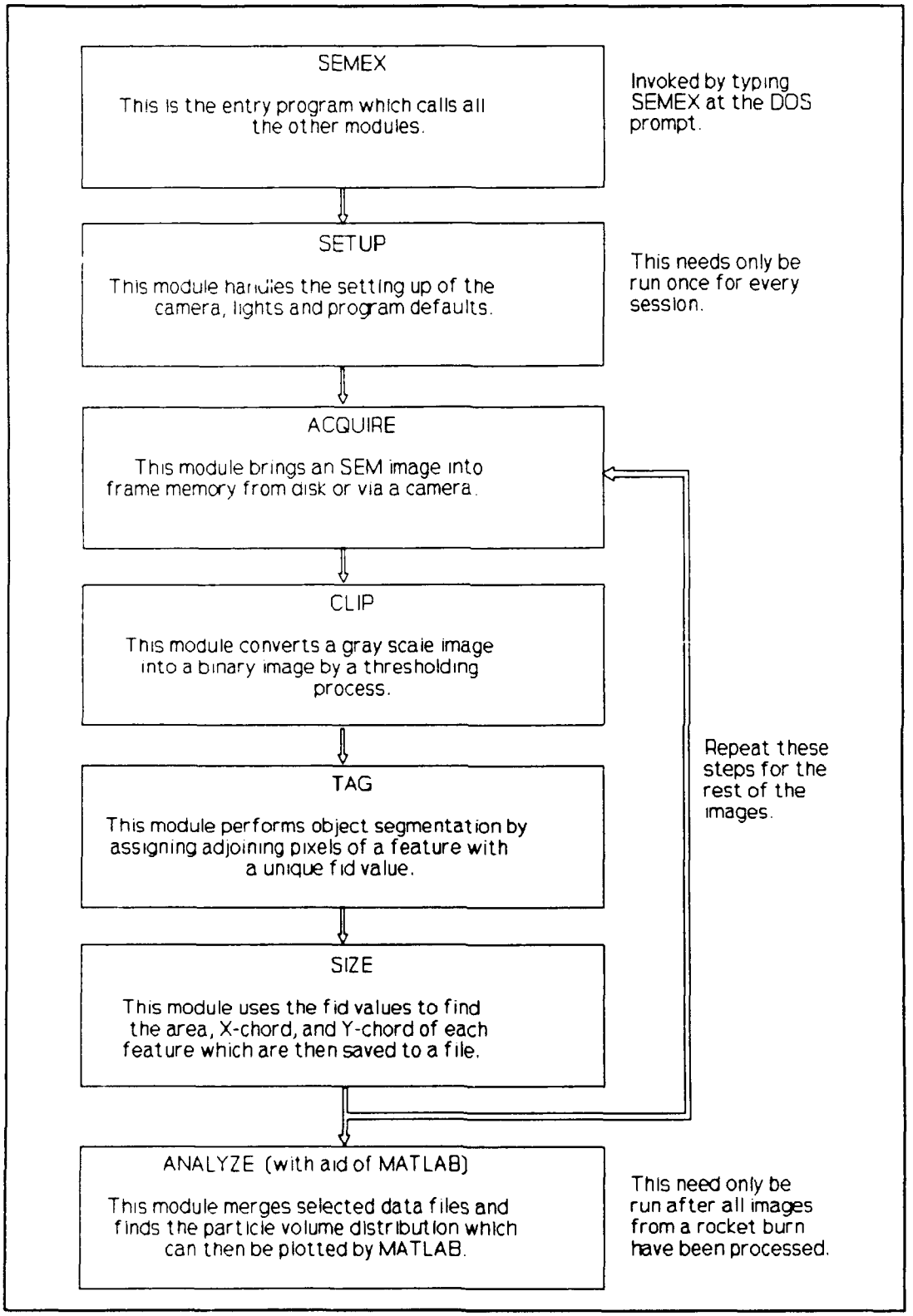

Figure 5. Typical sequence of activities in SEMEX.

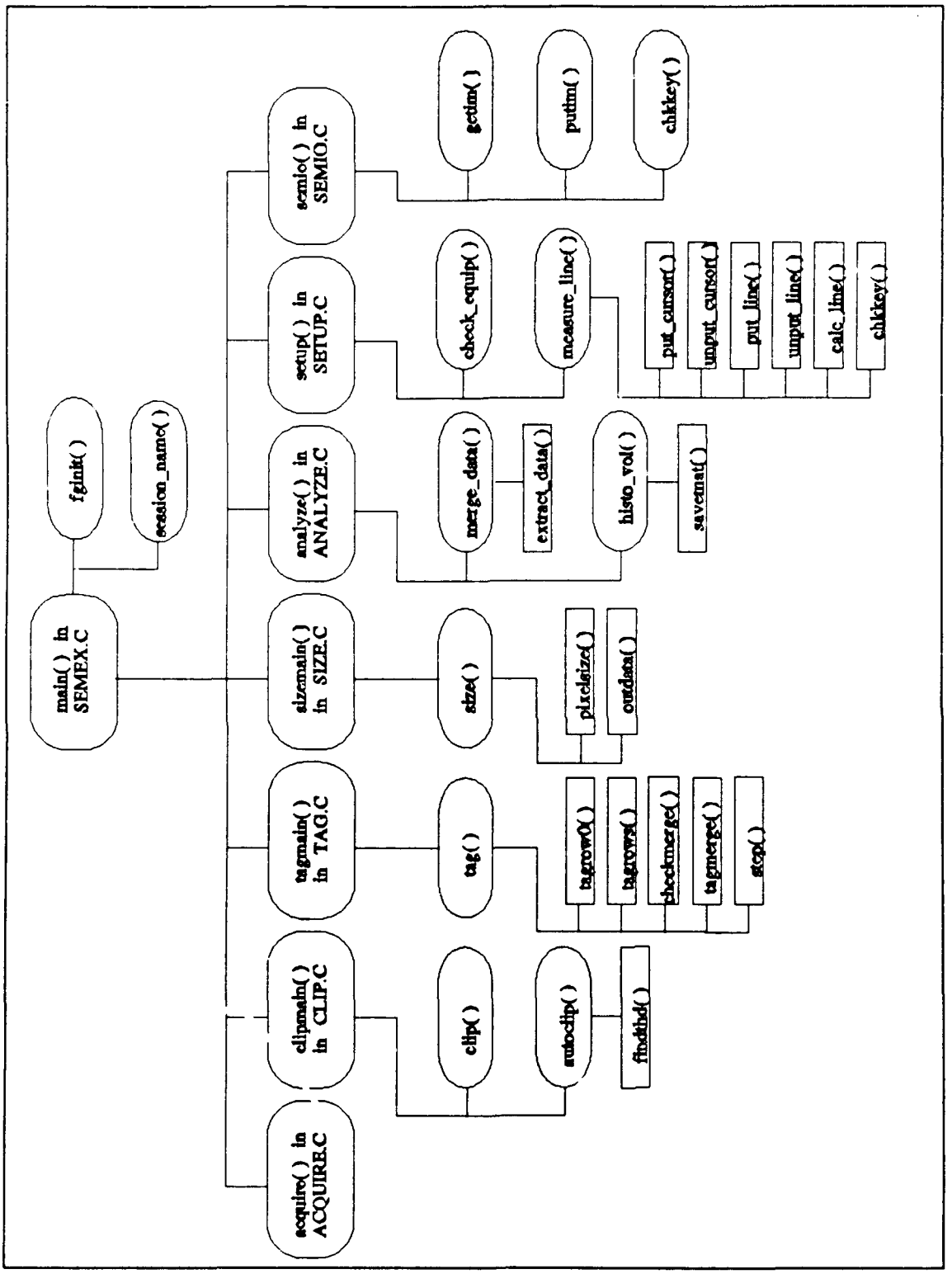

Figure 6. Block diagram of SEMEX modules (rounded rectangles) and their functions.

Figure 6 gives an overview of the various modules and functions, and the filenames they are contained in. Detailed descriptions of each of the program modules, the algorithms used and any trade-offs made follow.

A. PROGRAM MODULES

1. Module *MAIN*

The *MAIN* module consists of three functions *main()*, *fginit()* and *session_name()*. Figure 7 gives the algorithms, written in pseudo-code, for the

```
main()
{
    initialize frame grabber by calling fginit();
    blank out image filename;
    call session_name() to create sessions filename from current date;
    display popup menu of SEMEX options;
        accept user response and call respective functions;
        return to display popup menu again;
    update sessions file with elapsed time;
    display sign-off message;
}

fginit()
{
    set hardware definition;
    set frame dimensions;
    turn frame grabber on;
    initialize registers and LUTs;
    select camera input;
    clear frame memory;
    select and display frame memory;
    select input and output LUTs;
    return to main();
}
```

```
session_name()
{
    call dos_getdate() to get current date;
    call dos_gettime() to get current time;
    convert month and day to filename string;
    append file extension ".ses" to string;
    open session file using this string;
        record current date and time;
    close session file;
    return to main();
}
```

Figure 7. Pseudo-codes for SEMEX main program *main()* (top), frame grabber initialization *fginit()* (bottom left), and *sessions* file naming function *session_name()* (bottom right).

functions in this module. All C programs must begin execution with the function *main()*. It ties together all the other modules and functions in a program, and this is no different for SEMEX. In addition, it initializes all the global variables to their

default values. These global variables are used by the various program modules for range checking and for customizing the SEMEX program to suit user needs.

The function *fginit()* initializes the frame grabber by loading the correct addresses and configuration data. Once done, the frame memory is cleared and the input and output LUTs selected to prepare the frame grabber to receive digitized images captured with the vidicon camera. Function *session_name()* handles the formation of the *sessions* filename using the current month and day from the DOS system (obtained by calling *dos_getdate()* function), and appends a *.ses* file extension to it. However, the user can specify a different filename when inside the module *SETUP*. The *sessions* file records all the session activities for that particular day, regardless of the number of sessions. Figure 8 shows a typical *sessions* file.

```
Opening Session on 26 Feb 1991 at 18:09.
CLIP:  Image from Filename: sem02          .img
    Auto Threshold: 187      User Threshold: 187
TAG:  273 features tagged in 14.2 seconds
SIZE:  Vertical scale used: 8.400000 pixels/unit length
    Min Length spec:    1    Max Length spec:  100
    Sized 273 Features within specifications
    Sizing took 10.4 seconds
    Conversion constants: Cx=0.141691  Cy=0.119048  Ca=0.016868
            AREA_M      X-chord      Y-chord
  Max        5.803       12.044        4.167
  Min        0.017        0.142        0.119

ANALYZE:  Merging data files
                Extracting data from SEM02.DAT
                274 extracted.  Running Total is 273
                Total volume of minuscule particles is 3.726466 or   5.25%
                Printing results to Feb26.his
                Printing MAT-file sem02.mat
SEMEX was on for 2.4 minutes.
```

Figure 8. A typical session recorded in the *sessions* file.

2. Module *ACQUIRE*

The *ACQUIRE* module is actually a single function *acquire()* whose pseudo-code is shown in Figure 9. It accepts one of two sources of image input. The first makes use of a previously digitized image stored on disk. The second allows for

```
acquire()
{
    open dialog box;
    if disk image desired
        call SEMIO function getim();
    else
        display live video;
        if default selected
            use default
        else
            interac' / ·· adjust gain;
        if default se ected
            use default offset;
        else
            nteractively adjust offset;
    snap a single video frame;
    tran fer synchronization to frame grabber;
    de elect camera to prevent interference;
    rop left and right margins;
    complement digitized image if desired by calling ITEX PCplus function complement();
    if image is to be saved
        append gain, offset and margins to comments;
        call SEMIO function putim();
    update sessions file;
    close dialog box;
    return to main();
}
```

Figure 9. Pseudo-code for image acquisition function *acquire()*.

acquisition of a live image from the vidicon camera. If *SETUP* has not been run, *acquire()* will allow the gain and offset of the camera input to be interactively adjusted to obtain a high contrast image. After positioning the object image in the field of view of the camera, a single frame can be acquired (a process called snapping a frame by Imaging Technology). The camera input is then disconnected by software to reduce the jitter (caused by synchronization conflicts between the vidicon camera and the frame grabber) which can corrupt the digitized image. The

acquired image is 512 pixels wide by 480 pixels high with each pixel represented by 8 bits, thus allowing for 256 shades of gray.

Once acquired, the digitized image can have its left and right margins cropped. This will eliminate any edge effects and textual information that could interfere with the extraction of particle data. Next, the image can be complemented to produce a final image where particle features are dark and the background light. This is carried out by an ITEX PC*plus* function *complement()* which performs a one's complement (one bits become zeros and vice versa) on every pixel in the image.

Finally, the image can be saved to disk or left in the frame grabber's memory (frame memory) for further processing. If the image save option is exercised, program control is transferred to the *putim()* function in module *SEMIO* where a filename is requested. An extension of *.img* is automatically appended. The user will be asked to supply a line of comments which will be stored in the image header, together with information on the gain and offset of the camera input, margins set and the vertical scaling factor. This information will be displayed together with the digitized image whenever it is recalled by the other modules. The stored margins are used by the modules to define the area of interest in the digitized image. This can result in an improvement in performance, as the whole frame need not be processed. The vertical scaling factor provides the *SIZE* module with the conversion factor between pixels and some unit of measure (e.g., microns).

3. Module *CLIP*

The *CLIP* module consists of four functions namely, *clipmain()*, *clip()*, *autoclip()* and *findthd()*. Their pseudo-codes are shown in Figure 10.

```
clipmain()
{
    open dialog box;
    open session file;
    if disk image desired
        call SEMIO function getim();
    else
        use image in frame memory;
    call autoclip() to threshold the image;
    call clip() to modify the threshold;
    update sessions file;
    close sessions file;
    display options and get user response;
    if image to be restored
        linearize output LUT;
    else
        map output LUT into frame memory;
        if image to be saved to disk
            call SEMIO function putim();
    close dialog box;
    return to main();
}

findthd()
{
    within each region
        determine pixel value;
        if pixel value is more than mid-gray
            compare it to existing threshold,
            take the minimum of the two and
            assign it as the new threshold;
    return to autoclip();
}
```

```
clip()
{
    start repetition
        display clipped image;
        interactively get threshold;
    end repetition if user satisfied;
    supply threshold to calling program;
}


autoclip()
{
    define regions to be sampled;
    assume threshold is peak white initially;
    for each region
        call findthd() to find new threshold;
    use the minimum threshold for all regions;
    return to clipmain();
}
```

Figure 10. Pseudo-codes for *clipmain()*, interactive clipping *clip()* (top right), automatic clipping *autoclip()* (center right), and background threshold determination *findthd()*.

Function *clipmain()* is called by *main()* and uses the other functions in this module to convert the digitized gray scale image to a binary two-tone image by a process called thresholding. The gray scale image can be retrieved from the disk, or the existing image in the frame memory, acquired previously, can be used. The

IEEE Standard Glossary of Image Processing and Pattern Recognition Terminology (IEEE Std. 610.4-1990) defines thresholding as

> The process of producing a binary image from a gray scale image by assigning each output pixel the value 1 if its corresponding input pixel is at or above a specified gray level (the threshold) and the value 0 if the input pixel is below that threshold.

In mathematical terms, for a threshold T, the process can be represented as

$$p(i,j) = \begin{cases} 0 & p(i,j) < T, \\ 255 & p(i,j) \geq T. \end{cases} \quad (1)$$

where p(i, j) represents the gray scale value of the pixel at column i and row j. The gray scale value 0 represents peak black while the value 255 represents peak white.

The purpose of thresholding is thus to produce a binary image where the features are black against a white background. This is critical to the subsequent stages of processing. To ease the user workload, *autoclip()* and *findthd()* are called to automatically determine an estimate of the optimal threshold level. The algorithm assumes that the particle-to-background contrast is good and that the particles are dark against a light background. Given these two conditions, the algorithm examines six small regions around the edges of the image and one more in the center to determine the background gray scale levels. The location and size of the regions have been determined empirically, and is based on the location of the lighting. Pixels with gray scale levels lower than mid-gray (pixel value of 128) will be skipped over as they are assumed to be part of a particle and not the background. The darkest of the background levels obtained, corresponding to the lowest gray scale value for

all the regions, is used as the threshold. Sampling over several regions helps to account for any non-uniformity in illumination. This algorithm tries to ensure that background pixels do not accidentally become converted to particle pixels. While the algorithm is not robust, it was found to work satisfactorily for images satisfying the above two conditions. The processing time required to determine the automatic threshold level is less than one second. Once the threshold level is determined, a ITEX PC*plus* library function, *threshold()*, is called to perform the actual thresholding process.

As the automatic thresholding algorithm assumes *a priori* knowledge of the image, manual intervention is allowed using the *clip()* function. This allows the current threshold to be interactively changed to fine tune the image. Changes made to the image are constantly updated to the video monitor.

The resulting two-tone image can either be saved to a disk file for later processing, carried forward to the next stage of processing, or reverted to its original gray scale form to start over. Clipped files have a default file extension of *.iml*. Typically, the image is not saved but is kept in frame memory. A *SETUP* option allows the subsequent stages to be activated automatically to reduce user intervention and increase throughput.

4. Module *TAG*

The *TAG* module consists of the seven functions shown in Figure 6 earlier. Collectively, these functions determine whether a particle is made up of an isolated pixel or a group of adjacent pixels. The general term for this process of screening

the image for objects (features or particles) is called image or object segmentation. The IEEE Std. 610.4-1990 defines image segmentation as

> The process of dividing an image into regions for the purpose of object extraction.

```
tagmain()                                   tag()
{                                           {
    open a dialog box;                          determine maximum feature size limits;
    open sessions file;                         start timer;
    if disk image desired                       tag the first row of pixels by calling tagrow0();
        call SEMIO function getim();            tag subsequent rows by calling tagrows();
    else                                        determine total features tagged;
        use image in frame memory;              check search window size by calling checkmerge();
    tag image by calling tag();                 check for joined features by calling tagmerge();
    close sessions file;                        display final feature count;
    if tagged image to be saved                 display time elapsed;
        call SEMIO function putim();            update sessions file;
    close dialog box;                           update frame memory to make process permanent;
    return to main();                           return to tagmain();
}                                           }
```

Figure 11. Pseudo-code for *TAG* algorithm. Left column shows *tagmain()* while right column shows *tag()*.

Image segmentation or tagging, as it is subsequently referred to in this thesis, is performed in two stages. Figure 11 shows *tagmain()* calling *tag()* to scan the clipped image previously stored in the frame memory (or retrieved from a user-specified *.iml* disk file). Each non-background pixel is assigned with a feature identification number (fid). Isolated pixels are assigned unique numbers while adjacent pixels share the same number. Essentially, the fid number identifies pixels belonging to a single feature or particle.

The actual tagging is done by two functions *tagrow0()* and *tagrows()* which operate on the first row and the subsequent rows in the frame, respectively. Figure 12 shows the pseudo-code for the two functions. Essentially, the functions scan a row from left to right, skipping white pixels which are assumed to be part of the background. When a black pixel is detected, tne pixel above is examined. If that

```
tagrow0()
{
    for each pixel in the first row of the image
        skip to next pixel if present pixel is background;
        if left pixel occupied
            adopt its fid value;
        else must be new feature
            assign new fid value;
    return to tag();
}

tagrows()
}
    for each row of pixels in the image
        for each pixel in the row
            skip to next pixel if present pixel is background;
            if left or above pixel occupied
                adopt its fid value;
            else must be new feature
                assign new fid value;
            if subsequent adjacent pixels connected
                assign same fid value to these;
    return to tag();
}
```

Figure 12. Pseudo-codes for *tagrow0()* and *tagrows()* algorithms used for differentiating features from background in the first and subsequent rows, respectively.

pixel is not white, it already would have been assigned an fid value and the current black pixel is rewritten with the fid of the pixel above. In addition, the preceding pixel in the row is also checked and, if it has been assigned an fid value, this is also changed. This effectively identifies adjacent pixels as being part of the same feature, as long as the features are convex polygons (that is, there are no jagged edges caused by clusters of particles).

Because only 8 bits are allowed for storing a gray scale value, this allows for only 256 values. Allocating value 0 to peak black and value 255 to peak white leaves only 254 values. This means that only 254 particles can be uniquely identified. To overcome this limitation, the frame area is windowed into one or more rectangular regions. A simple function *checkmerge()* (see Figure 13) uses the initial number of features counted by *tagrow0()* and *tagrows()* to determine the dimensions

```
checkmerge()
{
    determine safe feature size within a group;
    if feature size is larger than safe size
        warn user;
        prevent larger features from being tagged;
    return to tag();
}

tagmerge()
{
    for each row in the image
        assign row search limits;
        for each pixel column in the image
            get pixel value of current pixel;
            get pixel value of pixel above;
            if either is background or
            if they are part of the same feature
                no action required, skip on to next column;
            else
                adjoining pixels have different fids and need to be merged;
            assign column search limits;
            for each row within the search limits
                for each pixel column within the search limits
                    if pixel is part of the current pixel
                        re-assign the value above to it;
    return to tag();
}
```

Figure 13. Pseudo-codes for *checkmerge()* and *tagmerge()* used for checking the number of features in a merging window and for carrying out the merging, respectively.

of this sizing window. The formula used is given by

$$\text{SAFESIZE} = \text{int}\left(\frac{480}{\text{int}\left(\frac{\text{fid}}{255}\right) + 2}\right) \qquad (2)$$

where SAFESIZE is the allowable size of the feature. The value 480 represents the number of rows in the frame and 255 is the number of unique fid values available plus one. The pseudo-function *int()* takes the integer portion of the quotient. For example, if the feature count is 254, the denominator will yield 2 and SAFESIZE would evaluate to 240 which is half the frame height. The sizing window is twice this value which implies that the whole frame is used. Features having a vertical length greater than SAFESIZE will be partitioned into two. If this happens, the program

will warn the user to rerun the *TAG* module after specifying that these larger features be excluded. Alternatively, the user may attempt to use a lower threshold level to reduce the number of features if the image is suspected to be noisy or the background is heavily textured. For any image, the maximum feature size should not be greater than a quarter of the image frame area (i.e., 256 x 240 pixels). Features within different regions may have the same fid number because, in addition, they are assigned a different group identification (gid) number to distinguish between them.

After the first pass described in the preceding paragraphs, another pass is taken through the whole image to merge joined features. This is carried out by *tagmerge()* and is necessary because the first pass assumes that all features are convex polygons. However, if some of the particles are overlapping or have irregular cross-sections, then they would be assigned different fid numbers although they are joined. Consequently, these non-convex polygons have to be merged together and assigned a common fid number. The number of features merged will be highlighted to the user together with the number of features identified or tagged. The latter can be used for verification of the *TAG* algorithm with the *SIZE* module described in the next sub-section. Both should yield the same feature count.

A limitation of the present algorithm is that border regions are ignored. This means that particles truncated by the edge of the image are treated as if they were complete particles. This may distort the actual distribution of particle sizes. This generally affects a small number of particles and is not expected to give rise to large errors in the result. However, should this error be deemed significant, the

usual practice is to define an area of interest somewhat smaller than the image such that a border exists. The border must be as wide as the largest particle. Particles lying on this border, and not truncated by the edge of the frame, will be tagged while those outside the border will be reset to the background color. In this way during *SIZE,* truncated particles will not be sized and hence do not affect the distribution. While this would yield more accurate results in terms of classifying particles, the smaller area of interest would yield a smaller sample population and more images would have to be used to compensate for this. In addition, it would not be practical to incorporate a border when particles could be as large as a quarter of the image.

The tagged image can be saved to disk. It will have a default file extension of *.im2.*

5. Module *SIZE*

The *SIZE* module consists of four major functions namely *sizemain(), size(), pixelsize()*, and *outdata()*. Collectively, they perform the function of determining the area and physical dimensions of the particles. The main function *sizemain()* allows for a stored tagged image to be retrieved, calls *size()* to coordinate the sizing process, and, finally, calls *outdata()* to tabulate and save the results. The pseudo-codes for *sizemain()* and *size()* are shown in Figure 14.

The function *size()* dynamically allocates storage arrays for particle areas, horizontal particle lengths (x-chords), and vertical particle lengths (y-chords). These arrays are indexed by the fid value of the feature whose dimensions are being stored. The function also keeps track of the processing time taken to size the image. Actual

```
sizemain()
{
    open a dialog box;
    open sessions file;
    if disk image desired
        call SEMIO function getim();
    else
        use image in frame memory;
    size image by calling size();
    if results are to be saved
        call outdata();
    close sessions file;
    if sized image to be saved
        call SEMIO function putim();
    close dialog box;
    return to main();
}
```

```
size()
{
    determine scale factor;
    determine feature size limits;
    check for sufficient memory;
    start timer;
    size the image with pixelsize();
    display time elapsed;
    return to sizemain();
}
```

Figure 14. Pseudo-codes for *SIZE* module functions *sizemain()* (left) and *size()* (right).

sizing begins when *size()* calls function *pixelsize()* to carry out the algorithm given in Figure 15.

Starting with the top left corner of the image, each row of pixels is scanned for the fid numbers previously assigned by *tag()*. Three registers are used for tracking the current feature area, feature x-chord, and y-chord. These are initialized to zero whenever a feature is first found. When pixels are found with an fid value corresponding to the current feature being sized, these pixels are reset to peak black to prevent them from being recounted. At the same time, the area and chord registers are updated. When the end of a feature is detected (no subsequent rows having the same fid value as the current feature being sized), these registers are stored into the respective storage arrays for later retrieval by *outdata()*. Statistics like the largest and smallest features are also determined, together with any features being rejected for failing to satisfy the size limits specified during setup. These size restrictions can be used to filter out background noise or undesirably large features.

```
pixelsize()
{
    for each row in the image
        assign row search limits;
        for each pixel column in the image
            get pixel value of current pixel;
            if pixel value is background or has already been sized
                skip on to next column;
            else
                pixel is part of a feature yet to be sized;
            assign column search limits;
            for each row within the search limits
                for each pixel column within the search limits
                    if pixel is part of the current feature
                        increment feature area;
                        increment horizontal feature length;
                        tag the pixel as sized;
                if horizontal feature length is zero
                    end of feature is reached;
                    go size next feature;
                else
                    update maximum feature length;
                    reset horizontal feature length;
                    increment vertical feature length;
            save vertical feature length;
            save feature area;
            reset vertical feature length and area;
            determine min and max feature dimensions sized so far;
            determine smallest and largest feature sized so far;
    check if any size limits exceeded;
    display any rejects together with number of features sized;
    update sessions file;
    return to sizemain();
}
```

Figure 15. Pseudo-code for function *pixelsize()* which scans each pixel in the image and identifies it as part of a feature.

The above procedure takes place inside a sizing window determined during setup (or modified by *check_merge()* as described in the earlier sub-section). The window must be as large as the largest pixel. This sizing window is moved from left to right, top to bottom over the whole frame. If the sizing window is large, a considerable amount of computation time is required to scan the area in the sizing window for pixels belonging to a particular feature. A performance enhancement has been incorporated by detecting the end of the feature and aborting the search for more pixels belonging to this feature. For small features, this results in a significant improvement in performance, as is shown in the next chapter.

Up to this point, all processing has been in terms of pixel dimensions to avoid floating point operations. For the vidicon camera used, a pixel is not square but has an X:Y aspect ratio of 1.2:1. To convert the pixel dimensions to microns requires the vertical scale factor which was determined during setup. The conversion is done in *outdata()*, shown in Figure 16. It uses the stored pixel values of the x-chords and y-chords and the total pixel area of each feature, and converts these to microns and square microns respectively. The results are tabulated on the computer monitor and saved to a *.dat* data file for subsequent analysis.

```
outdata()
{
    open a dialog box;
    calculate conversion factors;
    request for data filename;
    open data file;
    display feature horizontal and vertical dimensions;
    display feature area;
    write the same results to data file;
    close data file;
    display min and max feature sizes for the whole image;
    display smallest and largest feature area for the whole image;
    update sessions file;
    close dialog box;
    return to sizemain();
}
```

Figure 16. Pseudo-code for function *outdata()* which tabulates the feature sizes and writes it to a data file.

6. Module *ANALYZE*

Many methods exist for analyzing particles which are normally distributed [Ref. 15]. However, few exist for multi-modal analysis. For this application, the algorithm is designed to collect all the data files specified by the user, to extract the area data from them, and then to calculate the equivalent diameter and volume of a sphere. The end result of *ANALYZE* is a histogram of the sample population.

These steps are accomplished by the functions *analyze()*, *merge_data()*, *extract_data()*, and *histo_vol()*. The equivalent spherical volumes are used because a large particle has a much greater impact on the plume characteristics than many small ones. Referring to Figure 17, *analyze()* opens a dialog box and the *sessions* file and calls the other three functions. The function *merge_data()* builds up a list of data filenames by prompting the user with the name of each data file in the current directory and asks whether each should be used. The function *extract_data()* then begins extracting the area from each of the data files. The user can choose between using the calculated area (AREA_C) or the measured area (AREA_M). The function *histo_vol()* takes the areas and calculates the equivalent spherical diameters and volumes. Although the particles may be irregular in shape, the use of equivalent spheres facilitates the plotting of histograms against a single size dimension (particle diameter). The function then outputs the data into a *.his* histogram data file. This file can be read out or printed with any ASCII text editor program.

An option allows the user to save the same histogram data into a MATLAB-compatible *.mat* file. This makes use of a function called *savemat()* provided by MATLAB and has been locally modified for SEMEX. The histogram data can be plotted using a MATLAB script file called *SEM.M*. This file reads in the *.mat* file and plots a histogram of percent of total volume against a logarithmic scale of particle diameter. The listing for *SEM.M* is found in Appendix B. To enable comparison with the Malvern MasterSizer [Ref. 8], the same upper and lower limits for each bin is used. The only difference is that SEMEX has 38 bins as against only

```
analyze()
{
    open a dialog box;
    open the sessions file;
    call merge_data() to merge data from different data files;
    call histo_vol() to calculate volume and histogram the result;
    close session file;
    close dialog box;
    return to main();
}

merge_data()
{
    use _dos_findfirst() to get first data file and its creation date;
    display file and date created and ask user whether to include this file;
    if user response is positive
        update session file with datafile name;
        allocate memory for the list;
        add filename to list;
    find the rest of the data files using _dos_findnext();
    repeat the above steps until no more data files found;
    allocate memory for data array;
    call extract_data() to extract area from data file;
}

extract_data()
{
    for each of the selected data files in the list
        open the data file;
            read area into data array;
            repeat until end of file;
        close data file;
    erase the list;
}

histo_vol()
{
    determine bin limits;
    allocate memory for volume and diameter arrays;
    for each particle in the data array
        calculate the equivalent diameter assuming a circle, given area;
        calculate the equivalent volume assuming a sphere, given area;
        accumulate total volume;
        sieve the volumes and collate into the correct bins;
    update sessions file with particles outside the defined sizes;
    print out results to histogram file;
    if desired, a MATLAB .mat file can be created for SEM.M to plot the histogram;
    de-allocate memory thus erasing all the arrays;
}
```

Figure 17. Pseudo-codes for the *ANALYZE* module showing, from top to bottom, *analyze()*, *merge_data()*, *extract_data()*, and *histo_vol()*.

31 for the Malvern MasterSizer. The latter can only size down to 0.5 μm while SEMEX has a resolution down to 0.125 μm. SEM.M allows the results from the Malvern MasterSizer to be simultaneously plotted for comparison. These plots can be seen in the next chapter.

7. Module *SETUP*

The module *SETUP* consists of two functions *check_equipment()* and *setup()*. The first helps the user to set up the camera and light fixtures (refer to Figure 18), while the second is used to customize the SEMEX program.

```
check_equipment()
{
    open dialog box;
    if changes required to equipment setup
        acquire live video;
        interactively set video input gain;
        interactively set offset;
        digitize a single video frame;
        synchronize to frame grabber for stability;
        deselect camera to prevent interference;
        call measure_line() to measure 5 micron reference line;
        repeat measurements as desired;
        complement image to get dark features on light background;
        update frame memory to record changes;
        call ITEX PCplus threshold() to threshold image at maximum level;
        call clip() to interactively threshold the image;
        repeat whole process if desired;
        open sessions file;
        update setup parameters;
        close sessions file;
    close dialog box;
    return to setup();
}
```

Figure 18. Pseudo-code for the function *check_equipment()* which aids in the physical setup of the camera and the lights.

To set up the camera and light table, the function *check_equipment()* turns on the frame grabber and begins acquiring live video through the camera. The user can manually focus the vidicon camera and set its aperture. At the same time, the gain and offset of the camera input can be interactively set, and the photographic image aligned. In order to fit the 4:3 frame aspect ratio of the camera, the SEM photograph has to be rotated onto its side such that the textual information is to the right. Having done this, a single frame of the image is digitized.

Once acquired, *measure_line()* is called to allow features on the digitized image to be repeatedly measured with the aid of a graphics cursor. The graphics

cursor, drawn and erased by *put_cursor()* and *unput_cursor()*, respectively, is moved around on the video monitor by using the arrow keys. The function *chkkey()* translates these keystrokes into x and y values which *put_line()* and *unput_line()* use for drawing and erasing lines. The lines are measured by *calc_line()* and these measured lengths are recorded in the *sessions* file. Since SEM images have a 5 μm reference line on the image, measurement of this line allows for size calibration of the system. The function *measure_line()* assumes that this reference line is vertical and proceeds to calculate the vertical scaling factor to be used for converting pixels to dimensional lengths. Figure 19 shows the pseudo-code for this function. The graphics cursor and line manipulation functions are shown in Figure 20.

```
measure_line()
{
    display graphic cursor using put_cursor();
    await pressing of cursor keys;
    remove graphic cursor using unput_cursor();
    decode key pressed using chkkey();
    if valid cursor key pressed
        display graphic cursor at new location;
    else
        repeat above sequence;
    await key pressed to get second position;
    remove graphic cursor using unput_cursor();
    decode key using chkkey();
    if valid cursor key pressed
        put graphic cursor at new location using put_cursor();
        draw line to new location using put_line();
        calculate length of line using calc_line();
        if not zero length
            calculate scaling for a 5 micron reference line;
        open sessions file;
        update sessions file;
        remove line using unput_line();
        remove graphic cursor using unput_cursor();
        close sessions file;
    else
        go back to wait for second position;
    return to check_equipment();
}
```

Figure 19. Pseudo-code for the function *measure_line()* used to determine the vertical scaling factor of a 5 μm line in the SEM image.

```
put_cursor()
{
    determine top- and left-most pixels, given the center;
    save pixel values under the graphic cursor;
    if center pixel value is greater than mid-gray
        draw black graphic cursor;
    else
        draw white graphic cursor;
    return to measure_line();
}

unput_cursor()
{
    determine top- and left-most pixels, given the center;
    restore original pixel values thus erasing the cursor;
    return to measure_line();
}

put_line()
{
    if center pixel value is greater than mid-gray
        use black pixels to draw line;
    else
        use white pixels to draw line;
    if horizontal line is longer than vertical
        save horizontal pixel values under the line;
        draw a horizontal line;
    else
        save vertical pixel values under the line;
        draw a vertical line;
    return to measure_line();
}

unput_line()
{
    if horizontal line is longer than vertical
        restore horizontal pixel values under the line;
    else
        restore vertical pixel values under the line;
    return to measure_line();
}

chkkey()
{
    test for valid cursor keys;
    if either arrow keys is pressed
        increment/decrement x or y respectively;
    if Home, End, PgUp or PgDn key is pressed
        step x or y respectively (step size is 10 by default);
    return to measure_line();
}

calc_line()
{
    if horizontal line is longer than vertical
        return length of horizontal line to measure_line();
    else
        return length of vertical line to measure_line();
}
```

Figure 20. Pseudo-codes for functions *put_cursor()* and *unput_cursor()* which manipulate the graphics cursor while functions *put_line()* and *unput_line()* deal with lines.

Next, the digitized image can be complemented (to produce an image where particles are dark against a light background) and clipped. Clipping is carried out within *SETUP* by calling *clip()*, described earlier. This causes pixels with the same or lower values to be displayed as black, thus allowing the user to visually gauge the uniformity of the background. Figure 21 shows two images with varying uniformity in illumination.

Figure 21. Samples of clipped images showing the non-uniformity of the illumination. Darkened areas have the same or lower pixel values.

The left image is poorer than the right due to the poorer location of the lights. It is important that the illumination be evenly distributed so that the background intensities do not differ significantly over the image. One way of ascertaining this is to use a blank sheet of paper in place of the SEM image. The digitized image is then clipped with the threshold at maximum (threshold value 255). This should yield an almost black image initially. Decreasing the threshold will cause increasing portions of the image to turn white. The background should turn from completely black to completely white in about 50 levels of gray. If it takes more than

this or if the blacks appear blotchy, the illumination is uneven and needs to be re-adjusted.

If the initial clipped image is not almost completely black, it indicates that the full dynamic range of the contrast has not been exploited. The offset may need to be changed and then this whole process repeated until the results are satisfactory.

The second function *setup()* in the *SETUP* module allows the user to configure the SEMEX system, thereby streamlining the SEM extraction process. Figure 22 shows the pseudo-code for *setup()*. Various default parameters are

```
setup()
{
    check camera and lighting by calling check_equipment();
    open dialog box;
    display current default settings;
    accept user changes;
    display new default settings;
    close dialog box;
    return to main();
}
```

Figure 22. Pseudo-code for the function *setup()* in the module *SETUP*.

displayed and can be modified to enhance the accuracy of the extraction. The processes are automated to reduce user workload. Several flags can be defined which cause program flow to be altered, thus changing the appearance of the program and the amount of user intervention. In addition, a *sessions* file can be specified to record session activities and to store intermediate results, thus relieving the user from the need to write notes. A permanent record of each session can thus be kept. Figure 23 shows the *SETUP* dialog box with all the default parameters which the user can alter to tailor the SEMEX program.

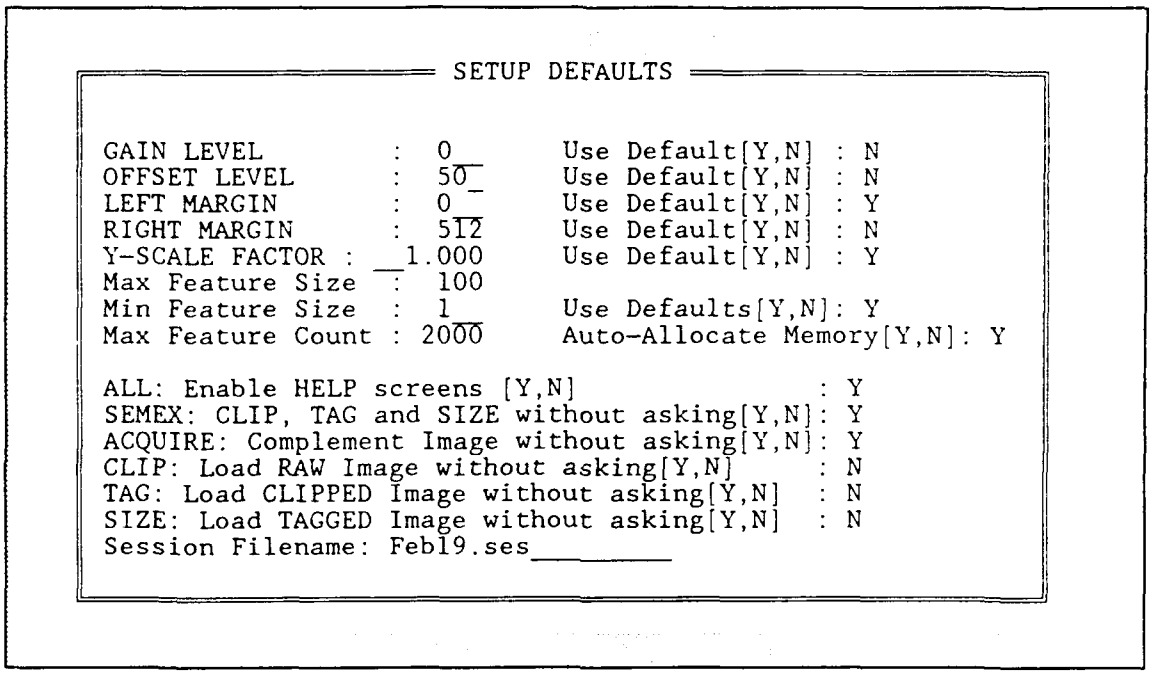

Figure 23. *SETUP* dialog box showing default parameters which the user can change to customize SEMEX.

8. Module *SEMIO*

The *SEMIO* module consists of several globally-used functions *getim()*, *putim()*, and *chkext()*, which are called by most of the other modules. It performs such functions as reading a disk image file into frame memory, writing out an image stored in frame memory to disk, and checking the filename extension. Image files are stored in a compressed format to reduce the disk storage space required for each image. However, this has a small penalty on the time required to read and write an image file (one or two seconds). Figure 24 shows the pseudo-codes for the three functions. The functions make use of two ITEX PC*plus* functions *readim()* and *saveim()* to actually perform the image read and save operations, respectively.

```
getim()
{
    call chgext() to append default file extension to filename;
    prompt user with filename;
    get user response;
    get image and comments from disk using ITEX PCplus library function readim();
    if error detected
        suggest trying again with new filename;
    display image on monitor;
    display comments in dialog box;
    extract margins and scaling factor from the comment line;
    return to calling program;
}

putim()
{
    call chgext() to append default file extension to filename;
    prompt user with filename;
    get user response and comments;
    append gain,offset, margin and scaling factor to comment string;
    save image and comments using ITEX PCplus library function saveim();
    if error detected
        suggest trying again with new filename;
    return to calling program();
}

chgext()
{
    determine the start of the file extension;
    discard old file extension;
    append default file extension;
    return to calling program;
}
```

Figure 24. Pseudo-codes for the *SEMIO* module. Functions *getim()* and *putim()* handle image transfers to and from disk while *chgext()* is used to set the default file extension.

The functions also handle the insertion of image parameters into the image header. Information like the gain, offset, margins and the scaling factor used in creating the digitized image are stored, together with any comments the user may wish to include. These parameter will be automatically retrieved when the image is brought in from disk. This ensures that no useful data is lost and it also reduces the setup time. Mundane tasks such as checking the existence of a filename and availability of disk storage space are also carried out by the functions in *SEMIO.*

V. RUNNING SEMEX

A. SEMEX SETUP PROCEDURE

SEM photographic images first have to be digitized and stored in the frame grabber's memory before any processing or extraction of information can be started. The camera and lighting setup is critical for obtaining a good digitized image. Using a 28 mm focal-length lens, the vidicon camera has to be mounted at a height of approximately 13 inches from the SEM photograph on the light table. A transparent glass plate is placed over the photograph to keep it flat. After ensuring that the video cables have been properly routed, the IBM AT is powered up and SEMEX started by typing SEMEX at the DOS prompt. A windowed menu like the one shown in Figure 25 will appear. Pressing a number will cause a corresponding selection to be highlighted. Alternatively, the arrow keys can be used to make a selection.

Once SEMEX has been started up, the first thing to do is to run *SETUP* by pressing [Enter] at the main menu prompt 1. Setup SEMEX. The setup procedure consists of five steps:

1. Initial alignment and focusing.
2. Setting camera input gain and offset.
3. Determining the system scaling factor.
4. Adjusting the illumination.

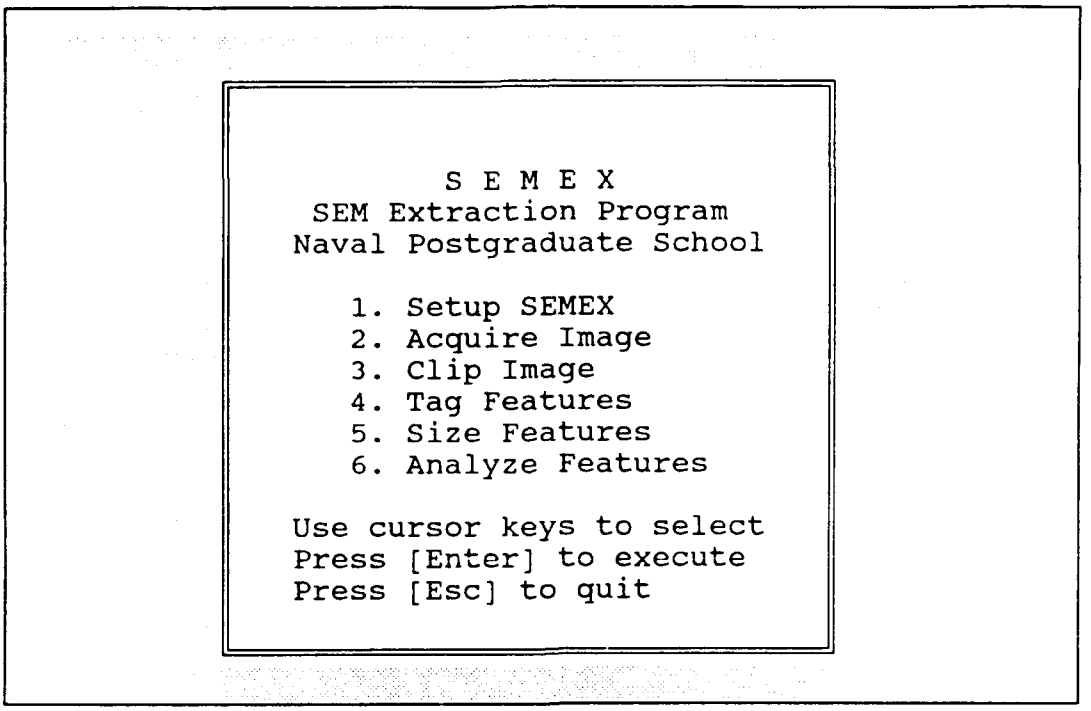

Figure 25. SEMEX main menu showing the six options.

5. Verifying and changing system defaults.

The detailed procedures are given in the following sections.

1. Initial Alignment and Focusing

Running *SETUP* causes another window to appear over the main SEMEX window. A prompt will appear asking whether the user wishes to set up the camera and lighting. Pressing any key other than 'N' or 'n' will select the default answer of 'Yes' and this turns on the camera. A digitized image of the SEM photograph can now be seen on the video monitor. Rotate the photograph counter-clockwise such that the aspect of the photograph is the same as the video monitor, with the textual information on the right. Adjust the height and focus of the camera so that the maximum area of the photograph can be seen without borders. The 5μm

reference line and part of the textual information on the photograph should also be visible on the right edge of the video monitor. Figure 26 shows the orientation of a typical image.

Figure 26. Digitized SEM image showing the orientation of the 5μm reference line and other textual information.

Ensure that the image is in focus. Once this is done for one photograph, the height and focus of the camera need not be adjusted again unless the setup is disturbed or dismantled. The photograph, the lens, and the glass plate should be dusted to prevent dust particles from being digitized.

2. Setting Camera Input Gain and Offset

The program will next ask the user to adjust the gain of the camera input; initially, the gain is set at maximum (value 0). The user can interactively alter this gain setting or, alternatively, adjust the aperture of the lens. A setting of gain value 0 at f/4 was found to be satisfactory. Pressing [Enter] moves the program on to adjust the offset. Again, the user can interactively change the offset or it can be

left at the default of 60. In general, this value should give the image on the video monitor good contrast.

3. Determining the System Scaling Factor

The vertical scaling factor can be determined by measuring the length of the 5 μm line located at the top right edge of the video monitor with the graphics cursor (a small cross-hair) located nearby. The cursor can be moved using the arrow keys for single pixel movements in either of four directions. For faster movement of the cursor, the [Home], [End], [PgUp], and [PgDn] keys can be used to move left, right, up, and down respectively, in steps of ten pixels. Pressing [Enter], when the cursor has arrived at one end of the 5 μm line, acquires the point. Next, move the cursor to the other end of the line and press [Enter]. This will cause the graphics line to disappear and the measured length in pixels will be displayed on the computer monitor, together with the vertical scaling factor (see Figure 27). If desired, other features can be measured. However, it should be noted that the scale factor from the latest measurement will be stored. This factor can also be changed in the *SETUP* screen.

4. Adjusting the Illumination

The placement of the lamps is crucial for obtaining even illumination. After visually determining that there is no glare or reflection from the lamps or overhead lighting, an image should be captured by pressing the spacebar. This causes the live video mode to be stopped and a single frame acquired. Press the [Enter] key if the image appears satisfactory or press the spacebar again to replace

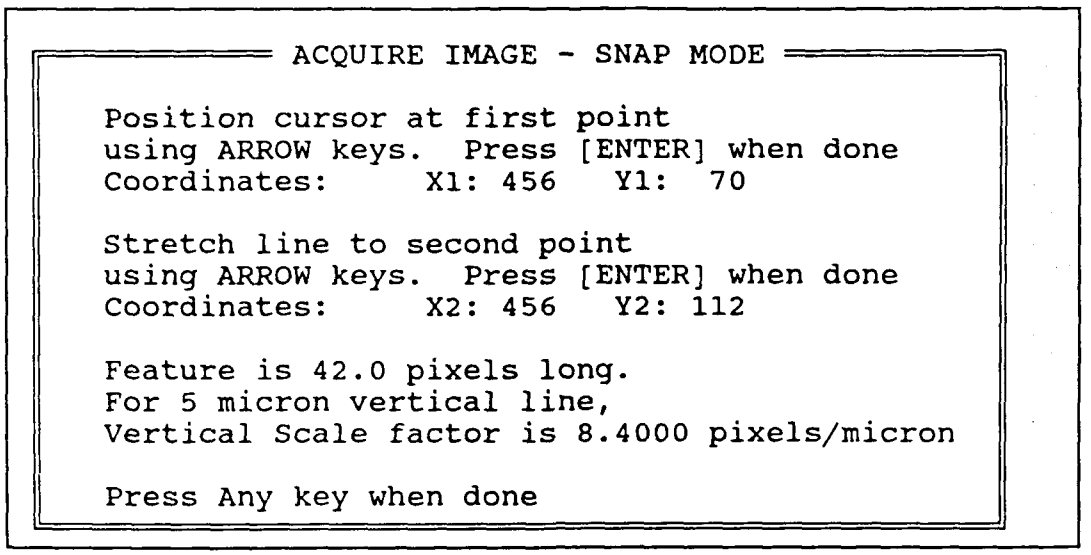

Figure 27. Measuring a line during *SETUP*. This determines the vertical scaling factor.

the old image with another digitized snapshot of the photograph. Next, complement the image if particles are white against a black background. The program then continues by clipping the image at threshold value 254 as shown in Figure 28.

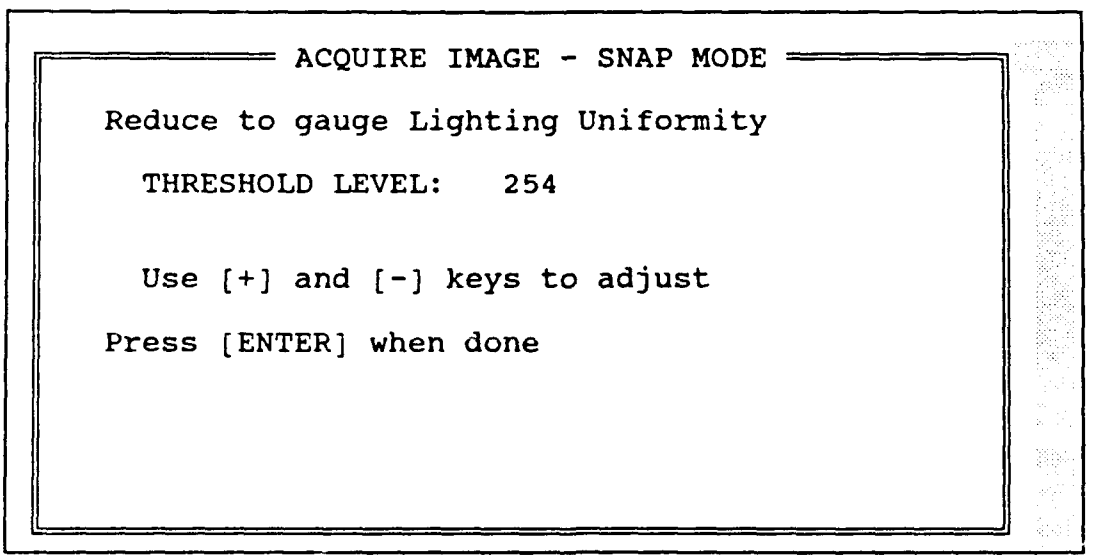

Figure 28. Adjusting the threshold level in *SETUP* to determine the uniformity of the illumination.

This should cause the whole image to appear almost black. If this is not the case, the offset may have to be changed. The user should use the [-] key to alter the threshold level until the background starts to turn white. If the photograph has been evenly illuminated, the background should turn white evenly. If the illumination is uneven, the background will appear blotchy. Press [Enter] to quit. This process can be repeated to re-adjust the lamp positions and camera input offset for optimal illumination and image contrast. An example of good even illumination, achievable with the current setup, is shown in Figure 26.

5. Changing System Defaults

The last step of *SETUP* is to verify and, if necessary, to allow the user to change the defaults that SEMEX and its modules will use. The defaults are shown in Figure 23. The user can sequence through the defaults, using the up and down arrow keys. If no changes are necessary, the user can simply press [Esc] at any point to terminate *SETUP* and return to the main menu.

B. IMAGE ACQUISITION PROCEDURE

Select 2 or move the cursor to 2. Acquire Image from the SEMEX main menu. If the gain and offset have been adjusted during setup, then there will be no prompts to adjust them here in *ACQUIRE*. If the user has elected to use a disk image, a dialog box like the one in Figure 29 will appear asking for the image filename. Upon retrieving the file, any comments embedded in the image header will be displayed. The program will prompt to allow the left and right margins to be

cropped by pressing the spacebar, so as to remove any artifacts or textual information (see Figure 30). Press the [Enter] key when done.

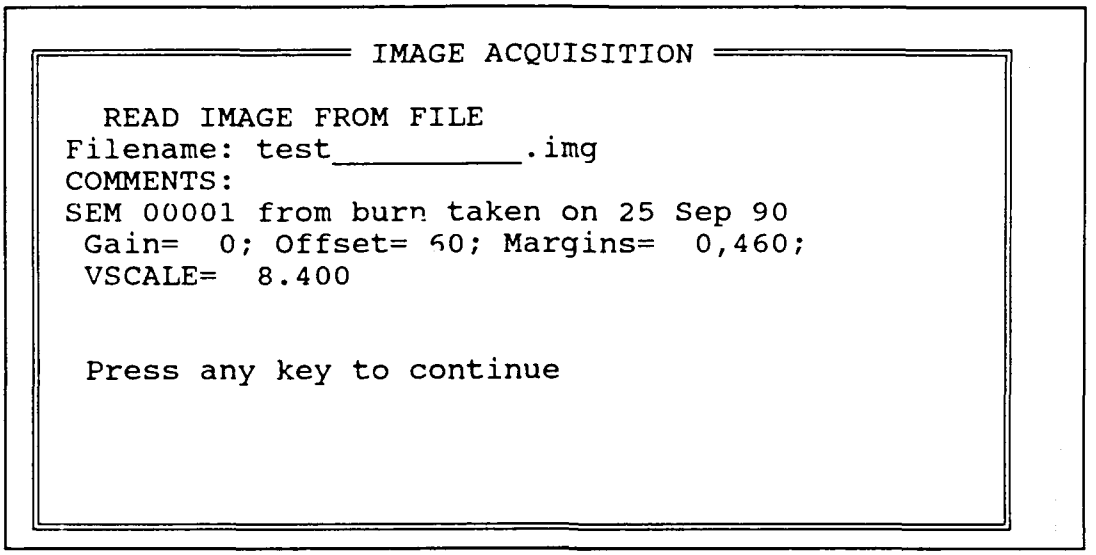

Figure 29. Acquiring a stored image into the frame memory. When the image is fetched from disk, the comments stored in the image header will be displayed.

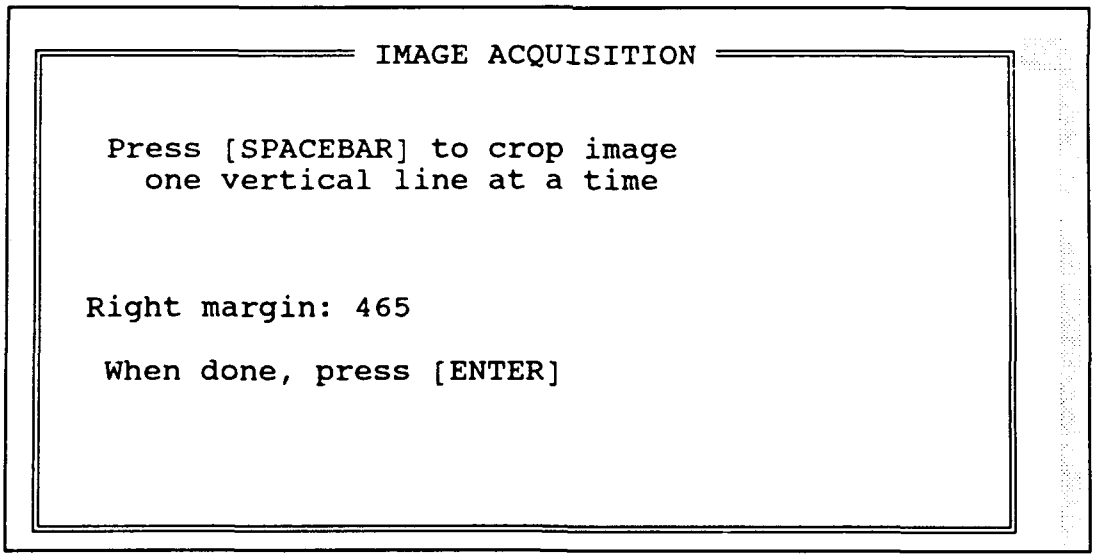

Figure 30. Cropping the right margin of the image in *ACQUIRE*. This is to remove the textual information on the image.

If the features appear white against a dark background, press [Enter] to complement the image, else type 'N' or 'n' to leave the image unchanged. The

user can then choose to save the image in a disk file or leave it in frame memory. Choosing the former will bring the user back to the SEMEX main menu after supplying the image filename and any comments. The program will automatically insert the image parameters (gain, offset, margins and scale factor) into the image header. The *sessions* file is also automatically updated.

C. IMAGE PROCESSING PROCEDURE

Image processing inside SEMEX consists of three steps:

1. A threshold is applied to the image to convert it into a binary image where particles are peak black and background pixels are peak white.

2. The image undergoes object segmentation whereby objects (i.e., particles) are discriminated from the background. Each object is given a unique identification number in the form of a gray scale value.

3. Each object is sized by measuring all pixels with the same identification number and converting to microns.

The details of these procedures are given below.

1. Thresholding with *CLIP*

The thresholding function is selected by typing 3 on the main menu which highlights the option 3. Clip Image. Pressing [Enter] activates the module *CLIP*. To threshold the digitized image already on the video monitor, the user simply presses [Enter] at the prompt 'Read from disk file [N]?'. The program then begins its automatic threshold level determination. Almost immediately, the clipped image is displayed on the video monitor. The user can choose to vary the threshold (to fine tune the thresholding process) by using the [-] or [+] keys (see Figure 31). The threshold value is displayed, as well as recorded in the *sessions* file. A copy of the

clipped image can also be saved to disk, although this is not necessary (see Figure 32). The default file extension is *.iml* for a clipped image.

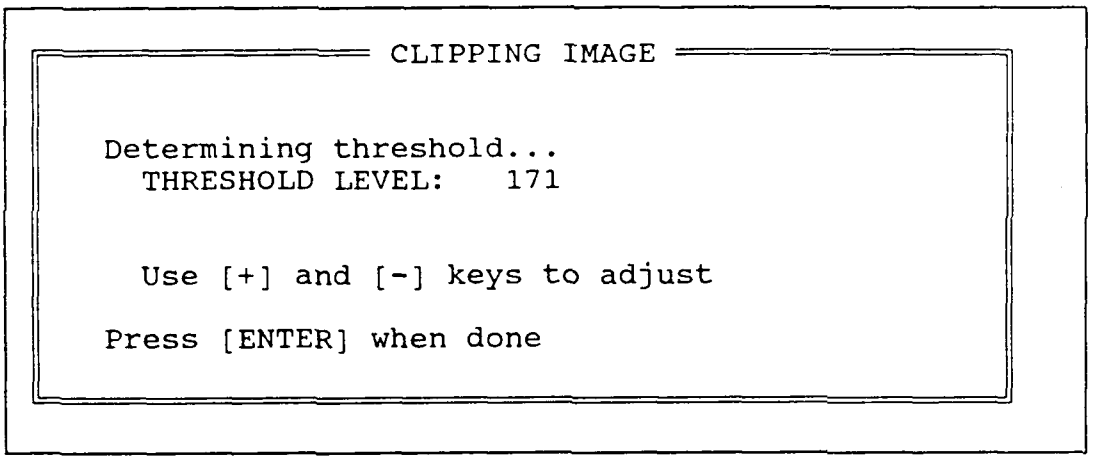

Figure 31. Clipping image using the automatic threshold which the user can subsequently change using the [+] and [-] keys.

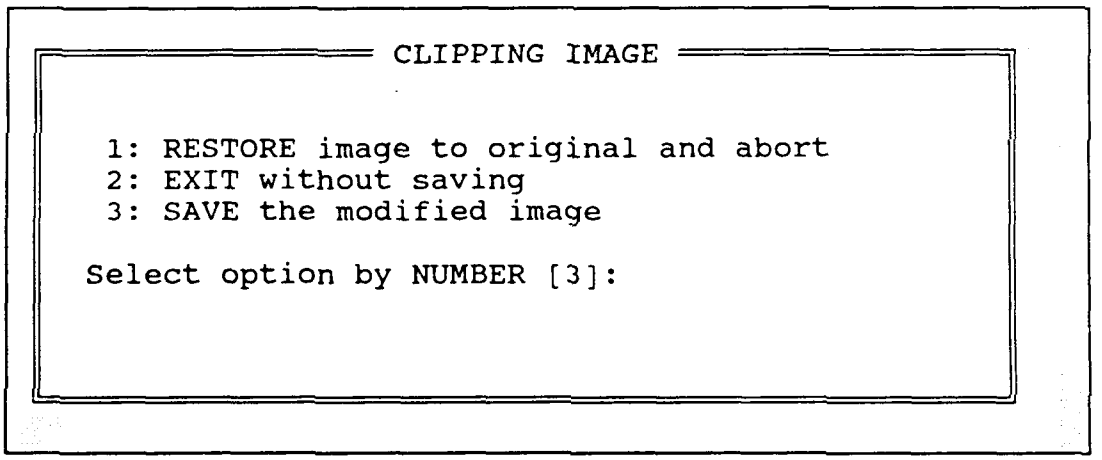

Figure 32. Screen for selecting the outcome of the clipped image. Option 3 is the default which can be invoked by pressing [Enter].

2. Image Segmentation using *TAG*

Image segmentation is performed by the *TAG* module. This module may be automatically invoked after *CLIP* completes, or it can be selected by pressing 4 or moving the cursor to 4. Tag Features. Features are tagged in two passes. The

first scans the clipped image and tags each pixel with an identification number (fid). Isolated pixels have unique numbers while adjacent pixels share the same number. Essentially, the fid number identifies pixels belonging to a single feature. The second pass searches for joined features with different fid numbers. These are merged, leaving only one fid number for all pixels belonging to that joined feature. Figure 33 shows the *TAG* screen after it has completed tagging an image.

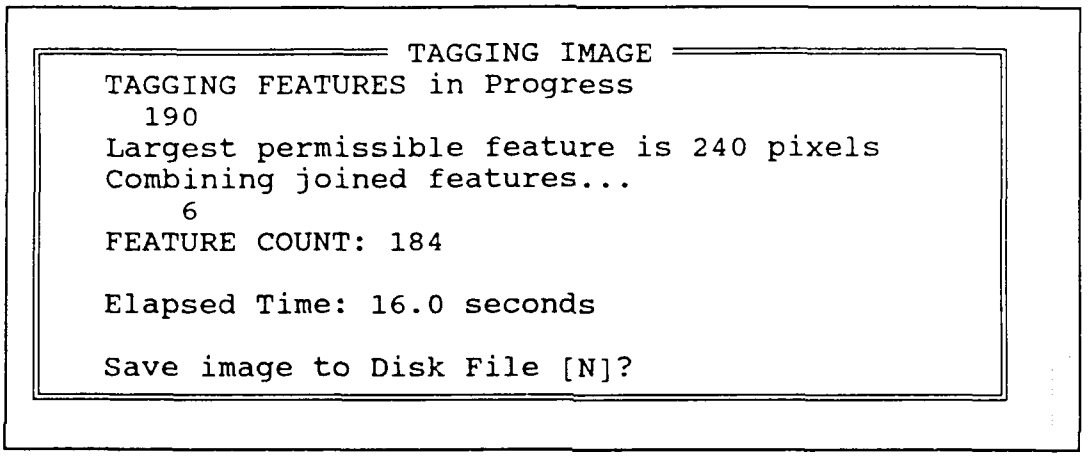

Figure 33. *TAG* screen showing the number of features, the number merged and the final count. The final prompt requests whether to save the tagged image.

3. Feature Sizing using *SIZE*

To run the *SIZE* option, the user selects 5 to highlight the option 5. Size Features. *SIZE* could be also automatically invoked after *CLIP* and *TAG*. *SIZE* will scan the tagged image in frame memory or on disk, and measure the pixel dimensions of each feature. If any feature fails to satisfy the size limits imposed during setup, these will be flagged out to the user. Figure 34 shows the information on the *SIZE* screen.

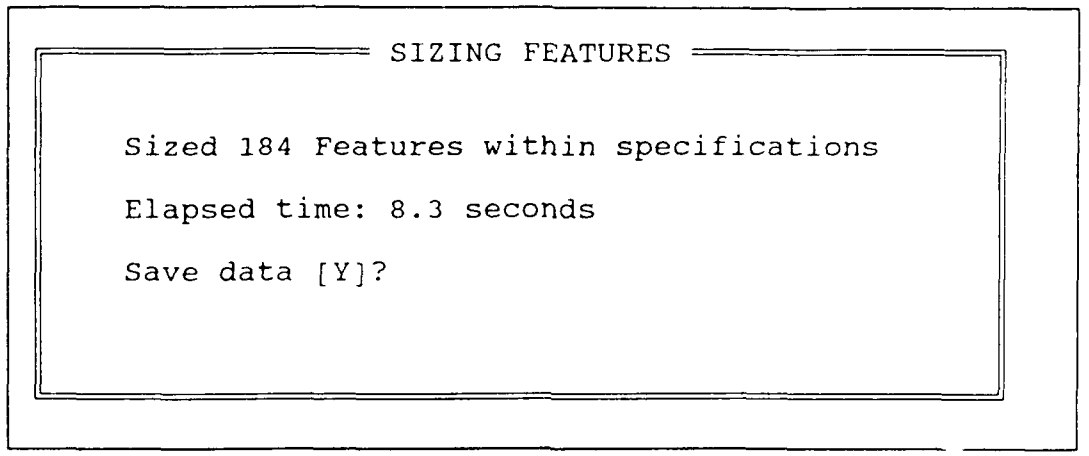

Figure 34. SIZE screen showing number of features sized and the time taken to size them. There is also a prompt to save the sized feature data.

If the sizing is successful, the user can save the results into a data file which has the same filename as the image but with the extension *.dat*. The results are also tabulated on the computer monitor twenty features at a time (see *Figure 35*). *The largest and smallest feature dimensions are also recorded in the sessions* file.

D. IMAGE ANALYSIS PROCEDURE

1. Merging the Data Files

Analysis of the images is usually carried out after a series of SEM images from the same firing have been sized. Each sized image will generate a data file containing areas and chord values. In order to obtain a statistically significant sample population and thereby reduce sampling errors, the data files from all the images belonging to one rocket motor burn have to be merged. Module *ANALYZE* handles this merging operation and is invoked by selecting 6 at the SEMEX main

TABLE OF FEATURE DATA

ID NO	AREA_C	AREA_M	X-Chord	Y-Chord
1	2.042	2.600	1.300	2.000
2	4.084	3.900	2.600	2.000
3	25.528	16.902	6.501	5.000
4	24.507	27.303	5.201	6.000
5	70.457	61.106	3.900	23.000
6	2.042	2.600	1.300	2.000
7	1.021	1.300	1.300	1.000
8	1.021	1.300	1.300	1.000
9	1.021	1.300	1.300	1.000
10	15.317	15.602	3.900	5.000
11	8.169	6.501	2.600	4.000
12	1.021	1.300	1.300	1.000
13	3.063	3.900	1.300	3.000
14	20.422	18.202	2.600	10.000
15	9.190	9.101	3.900	3.000
16	1.021	1.300	1.300	1.000
17	1.021	1.300	1.300	1.000
18	1.021	1.300	1.300	1.000
19	4.084	5.201	2.600	2.000
20	1.021	1.300	1.300	1.000

Press [ENTER] for MORE or [ESC] to QUIT

Figure 35. Tabulated data showing the calculated equivalent elliptical areas (AREA_C), the measured pixel areas (AREA_M), the X-Chords and Y-Chords for 20 features.

menu 6. Analyze Features. The module will begin by listing all the *.dat* data files in the current directory one at a time, for the user to select (see Figure 36).

2. Calculating Particle Volume

ANALYZE calculates the particle volume by assuming that the particles are spheres. It takes the measured particle areas from all the selected data files and determines the equivalent particle diameters and volumes. A 38-bin histogram is tabulated and displayed 10 bins at a time. This result is also saved into an ASCII *.his* file and can also be saved in a MATLAB-compatible *.mat* file. If the latter is chosen, the user can quit SEMEX and have the histogram plotted by running *SEM.M*

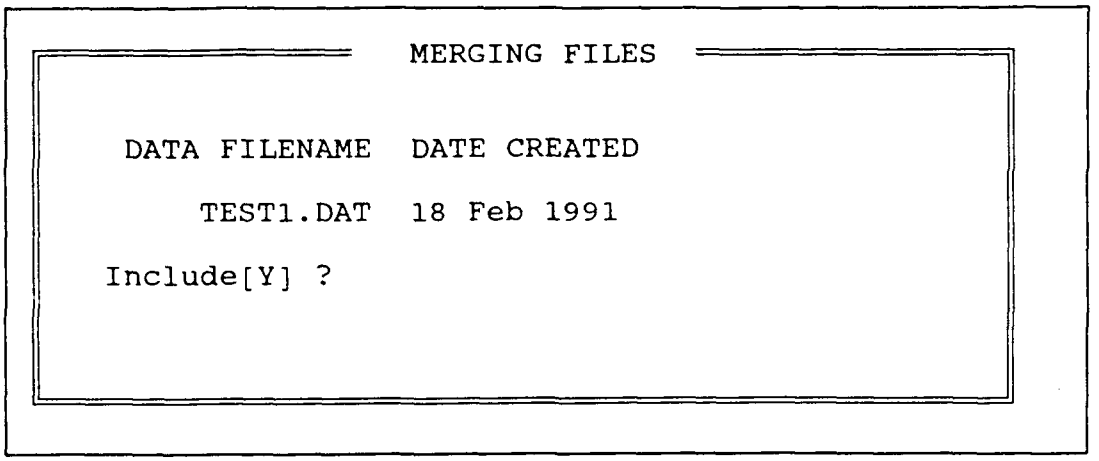

Figure 36. ANALYZE screen prompting one data file at a time with its date of creation. Pressing 'Y' or 'y' accepts the datafile. A count is kept of the number of files selected.

inside MATLAB. The histogram plot shows percentage of total particle volume against a logarithmic scale of particle diameter. This plot format is similar to that put out by the Malvern MasterSizer. For ease of comparison, both SEMEX and Malvern data can be plotted. Sample plots are shown in the next chapter.

VI. EXPERIMENTAL RESULTS

This chapter describes the preliminary experimental results obtained. It is divided into three sections: system calibration and error quantification, comparison of the performances of SEMEX and HOLOGRAM, and, correlation of the results with that of the Malvern MasterSizer.

A. CALIBRATION

1. Determining Pixel Aspect Ratio

In order for *outdata()* in *SIZE* to properly convert pixel counts into dimensional lengths and areas, there is a need to first determine the pixel aspect ratio. In the vidicon camera used, the aspect of a single pixel is not square. To determine the actual aspect ratio, a three-inch rule was placed, first horizontally, then vertically. The digitized lengths were then measured using the *measure_line()* function in *SETUP*. The measurements are tabulated in Table I.

Table I. CAMERA PIXEL SIZE

Orientation	Horizontal	Vertical
3-inch Reference length	350 pixels	420 pixels
Pixel Length (inch/pixel)	0.00857	0.00714

The pixel aspect ratio can be obtained by taking the ratio of the horizontal length to vertical length of a single pixel. This is equal to 420 : 350 (or 1.2) since the

same 3-inch length was measured. In SEMEX, this constant is defined as the ASPECT_RATIO. This works out to be the same as the frame aspect ratio.

2. Quantifying System Errors

To quantify the system errors, a test pattern was created by placing on a white background, 48 black etch-resistant circles used in printed circuit artwork (commonly called donut pads). Each circle has an outer diameter of 0.187 ± 0.003 inches and an inner diameter of 0.062 ± 0.003 inches (see Figure 37). The variability of the outer diameter gives rise to an error of 1.6%.

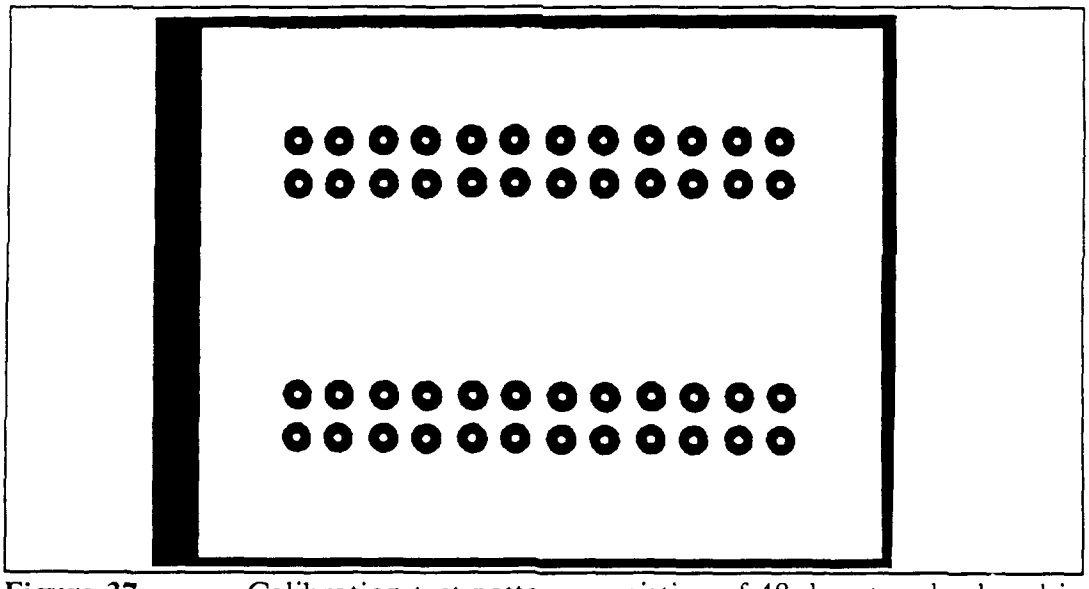

Figure 37. Calibration test pattern consisting of 48 donut pads placed in four rows.

The gain, offset and illumination were adjusted in accordance with the setup procedure and precautions given in Chapter V. The *measure_line()* function was used to determine the vertical outer diameter of the donut pad, in pixels. The vertical scale factor, VSCALE, could then be determined by taking the ratio of the measured pixel length (28 pixels) and the true vertical length (0.187"). For the setup

used, VSCALE was found to be 149.733 pixels/inch. Since *measure_line()* only determines VSCALE correctly for a 5 μm line, VSCALE has to be manually entered in the *SETUP* screen. SEMEX calculates the horizontal scaling factor by dividing VSCALE by ASPECT_RATIO. The area represented by a single pixel can then be found by taking the reciprocal of the product of the horizontal and vertical scaling factors. This area, multiplied by the number of pixels covering a particle, gives the measured area of that particle.

Next, the *ACQUIRE* module was executed to digitize the test pattern into frame memory. *CLIP* was then invoked and the automatic thresholding function *autoclip()* applied. This yielded a visually-clean binary image for a threshold of 157. After exiting the *CLIP* module, *TAG* and *SIZE* were invoked. The results are tabulated in Figure 38. The maximum percentage error is given by the maximum deviation from the mean expressed as a percentage of the mean.

The elliptical areas (labelled AREA_C in column 2) were calculated by taking the products of the x-chords and y-chords (these correspond to the major and minor axes of an equivalent ellipse) and multiplying by the constant, $\pi/4$. The AREA_C values corresponded reasonably well with the actual pixel areas measured (labelled AREA_M in column 3). The difference of the two mean areas amounted to 3.9% of the mean of AREA_M.

Although the effect of the non-square pixel aspect ratio had been compensated for, the results showed that the x-chord and y-chord lengths were still not equal. The difference between the two mean chords was 9.4% of the mean y-

ID	AREA_C	AREA_M	X_CHORD	Y_CHORD
1	0.029	0.028	0.184	0.200
2	0.029	0.028	0.184	0.200
3	0.026	0.025	0.168	0.194
4	0.026	0.026	0.168	0.194
5	0.027	0.027	0.176	0.194
6	0.028	0.027	0.184	0.194
7	0.028	0.027	0.184	0.194
8	0.027	0.027	0.176	0.194
9	0.027	0.026	0.176	0.194
10	0.024	0.023	0.160	0.194
11	0.024	0.024	0.160	0.194
12	0.027	0.026	0.176	0.194
13	0.028	0.027	0.184	0.194
14	0.027	0.025	0.176	0.194
15	0.027	0.026	0.176	0.194
16	0.027	0.026	0.176	0.194
17	0.026	0.026	0.176	0.187
18	0.027	0.026	0.176	0.194
19	0.027	0.026	0.176	0.194
20	0.027	0.026	0.176	0.194
21	0.024	0.023	0.160	0.194
22	0.025	0.024	0.168	0.187
23	0.026	0.025	0.168	0.194
24	0.024	0.022	0.160	0.187
25	0.025	0.023	0.168	0.187
26	0.026	0.024	0.168	0.194
27	0.027	0.025	0.176	0.194
28	0.027	0.026	0.176	0.194
29	0.027	0.026	0.176	0.194
30	0.027	0.026	0.176	0.194
31	0.028	0.027	0.184	0.194
32	0.028	0.027	0.184	0.194
33	0.027	0.026	0.176	0.194
34	0.027	0.025	0.176	0.194
35	0.024	0.024	0.160	0.187
36	0.025	0.024	0.168	0.187
37	0.024	0.023	0.160	0.187
38	0.024	0.024	0.160	0.187
39	0.025	0.025	0.168	0.187
40	0.026	0.025	0.176	0.187
41	0.026	0.026	0.176	0.187
42	0.028	0.027	0.184	0.194
43	0.027	0.026	0.184	0.187
44	0.028	0.026	0.184	0.194
45	0.028	0.026	0.184	0.194
46	0.027	0.026	0.176	0.194
47	0.026	0.025	0.176	0.187
48	0.027	0.025	0.176	0.194
Mean	0.027	0.026	0.174	0.192
Max % Error	9.43	13.73	8.15	3.71
Std Deviation	0.001	0.001	0.008	0.004

Figure 38. Output from *SIZE* showing the results obtained from the calibration test pattern. The calculated area, AREA_C, is given by $\pi/4$ * X_CHORD * Y_CHORD.

chord. The deviation within each chord amounted to a 8.2% error for the x-chord and a 3.7% error for the y-chord. Taking into account the variability of the donut pads (1.6%), the system has contributed to errors of 6.6% and 2.1% for the x-chords and y-chords respectively.

In an attempt to identify the source of this error, a square grid was digitized. A rectangular graphics box was then placed over the grid lines. It was found that vertical lines near the right edge were slanted while lines along the other three edges were correctly digitized (remained orthogonal). Figure 39 shows the

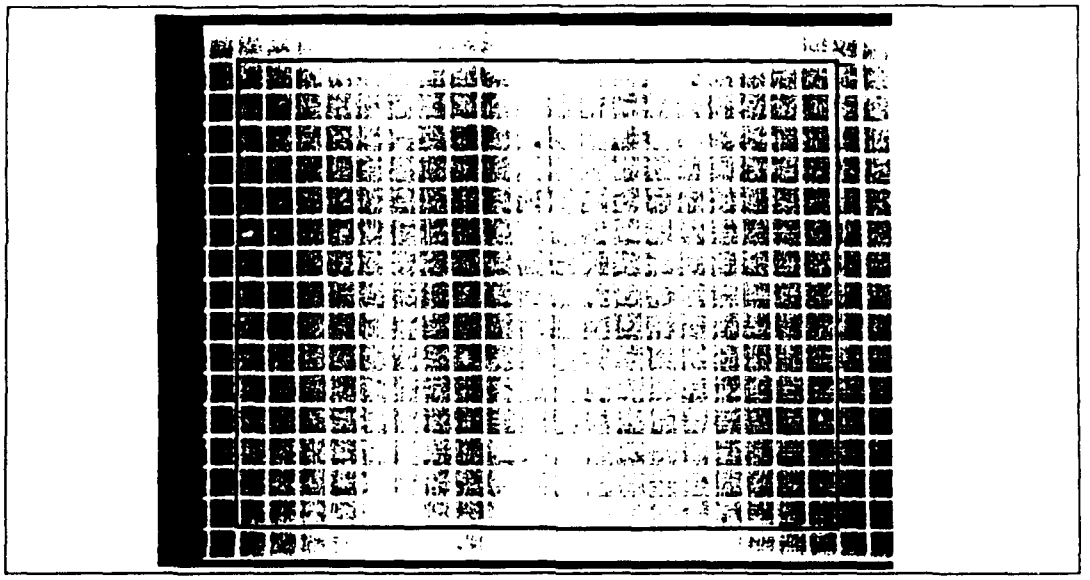

Figure 39. Digitized image of a square grid showing that vertical lines near the right edge were slanted when compared with the dark rectangular graphics box.

square grid image with the artifact enhanced by the rectangular box. The maximum divergence was found to be three pixels. Using a plumb line and a liquid level, the light table and camera were checked for squareness and were found to be accurately aligned. It is suspected that this error is due to variations in the horizontal scan

velocity of the vidicon camera down the frame. If this is true, then the vidicon camera has introduced a scanning error of three pixels. However, this amounts to only 0.6% error over the whole frame length and is insufficient to account for the deviation in the chord values.

Next, a 3-dimensional plot was made by sampling the pixel values around a single donut pad (32 x 32 pixels) in the calibration test pattern. Each grid square represents one pixel. The pixel values are represented by the height (z-axis) of the plot. Figure 40 shows that the edges of the donut pad are not vertical. This suggests

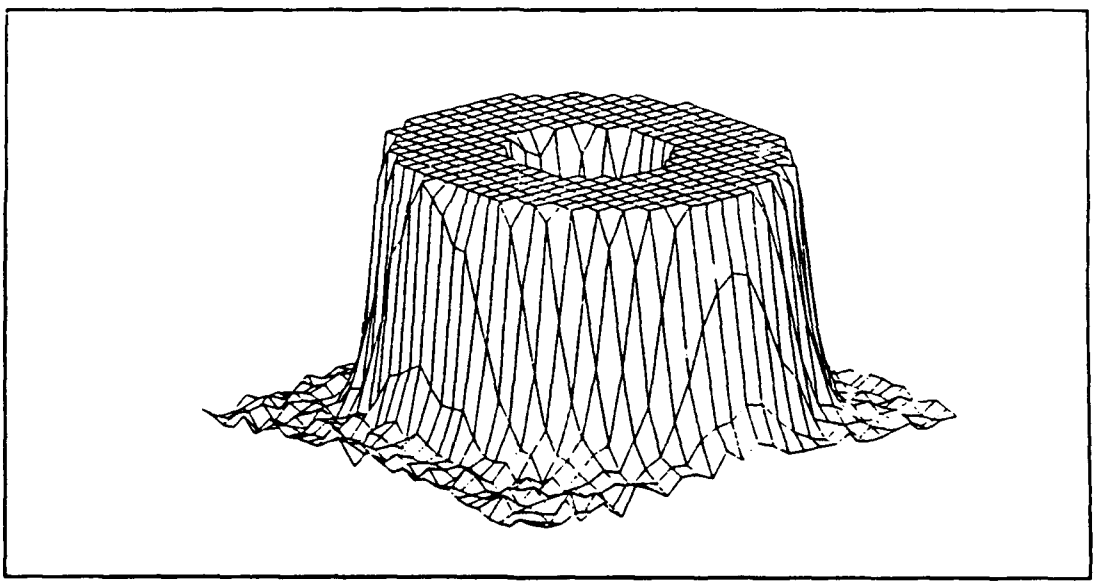

Figure 40. 3-D plot showing the pixel values from a digitized image of a donut pad. Note that the plot has been inverted for clarity. This gives rise to high peaks for dark regions.

that the analog-to-digital (A/D) converters in the frame grabber are not fast enough to digitize a sharp transition from peak white to peak black (the latter is represented by the flat circular region on the plot). The contrast transfer function of the camera and lens system also contributes to this limitation. The slope represents the amount

of error. From the plot, the change from peak white to peak black takes one to two pixels. As there are two slopes for each feature, the error could be as much as 4 pixels. Depending on the threshold selected, the cross-sectional area would change. The worst case slew rate error for the case of the donut pad would be 4 pixels out of 28 (the measured pixel length), or 14.3%. The error is a function of the size and contrast of the feature. This error is very significant for small features of the order of one to five pixels. For the magnification used in the SEM, a 1/8 μm particle would occupy only two pixels and have a slew rate error of up to 33.3%. As the particle size increases, the error drops dramatically. A 1 μm particle, for example, would have a maximum slew rate error of only 6.3%.

A major problem faced is the determination of the proper threshold to use. In chapter V, an automatic thresholding algorithm was described. The use of an automatic threshold can significantly improve the throughput. However, the criteria for threshold determination is an important consideration. Figure 41 shows a three-dimensional plot of a 32 x 32 pixel gray scale image. The left portion shows the varying gray scale levels of each pixel. When the region is clipped, the gray scale levels below the threshold are suppressed. However, different thresholds result in vastly different results as can be seen by the center and right portions of the figure. The right portion was obtained using the automatic thresholding algorithm. From the figure, it can be seen that the limited slew rate has prevented small features from exhibiting a high pixel value. Hence, they tend to be suppressed. In this particular case, the *autoclip()* function was able to bring out thirteen features.

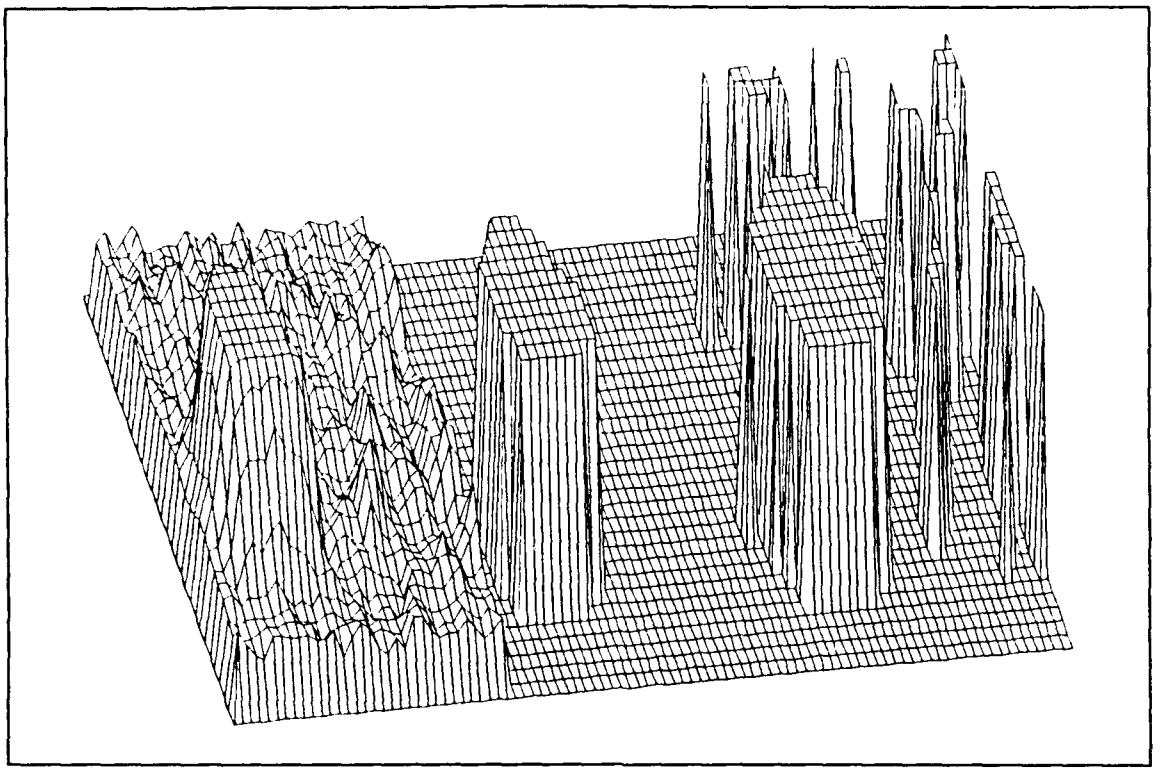

Figure 41. Three-dimensional plot showing a 32 x 32 pixel region drawn three times at different thresholds. The left portion is the gray scale image. The center and right portions are at thresholds of 33 and 165.

B. PERFORMANCE COMPARISON WITH HOLOGRAM PROGRAM

The HOLOGRAM program written by Hockgraver [Ref. 6] consists of the following modules:

1. Filtering routines.
2. Image threshold.
3. Feature identification.
4. Feature sizing.

Of these four modules, only the last three modules have similar functions in both HOLOGRAM and SEMEX. The first module (filtering routines) is used to reduce the effect of speckle introduced during image reconstruction from a hologram. This is not applicable to the SEM images as there is no speckle. In addition, SEMEX has two essential stages not carried out by HOLOGRAM. These are the acquire and analyze stages. HOLOGRAM depended on the ImageAction*plus* software to perform image acquisition and on Statgraphics for analysis. This generally took much longer, as the ImageAction*plus* was not tailored to perform the acquisition, cropping and complementing of the images in an efficient manner. Also, the scaling factors had to be manually determined. Similarly, Statgraphics required its own setup procedure.

Comparison of the performance of the three modules common to SEMEX and HOLOGRAM was carried out by subjecting both programs to a common set of images. These images were previously acquired and then stored on disk. The images had different numbers of features ranging from 48 to 920, and are shown in Figure 42. The execution times are tabulated in Table II and the speedup calculated. Speedup is defined as

$$\text{Speedup} = \frac{\text{Execution Time of Slow System (HOLOGRAM)}}{\text{Execution Time of Fast System (SEMEX)}} \quad (3)$$

and is a standard measure for the performance improvement of two systems.

From the results, it can be seen that the speedup becomes more significant as the images become more complex. Almost an order of magnitude improvement has

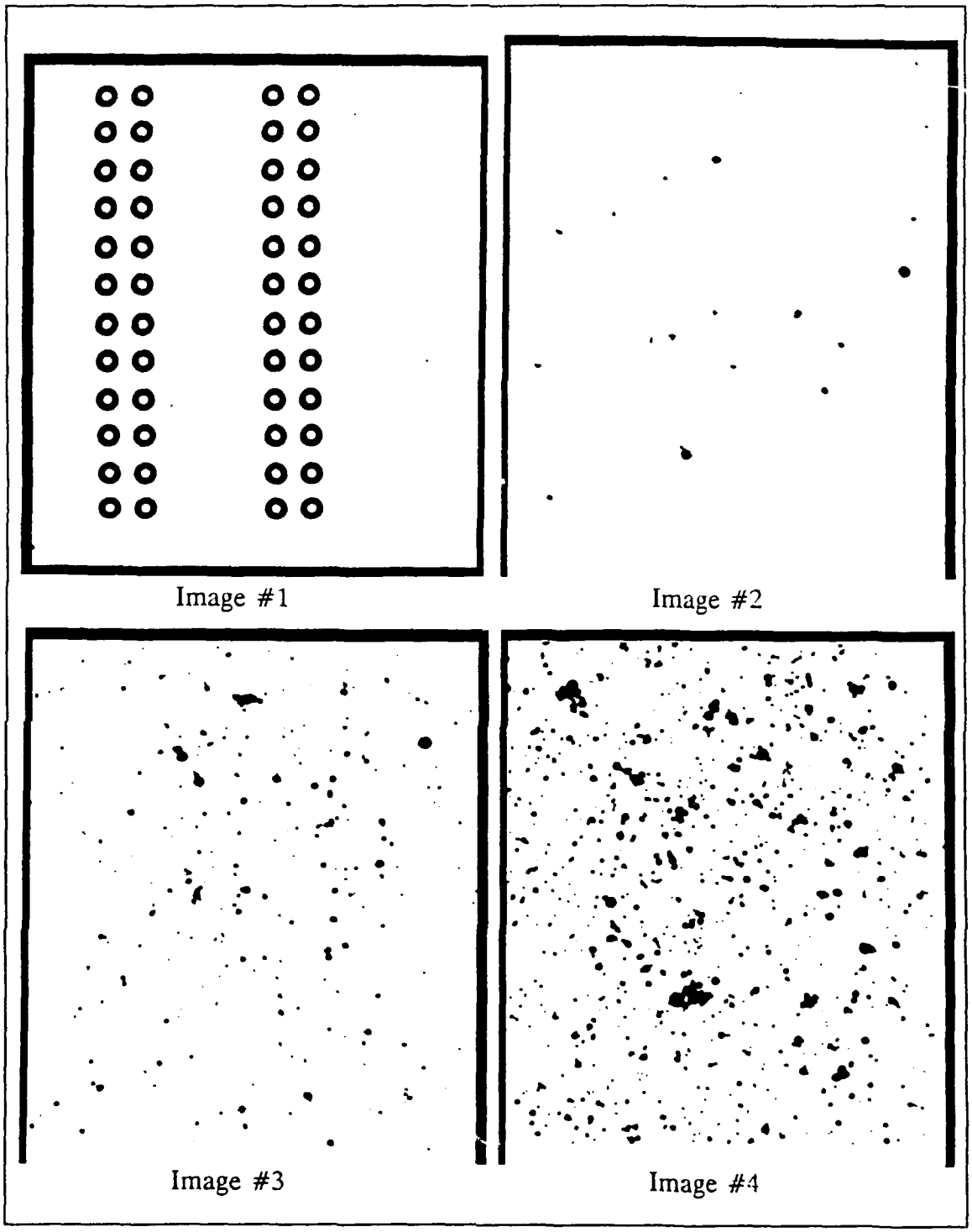

Figure 42. Images used for testing the performance of SEMEX against HOLOGRAM. Image #1 is the calibration test pattern with 48 features while images #2, #3 and #4 are actual SEM images with 177, 495 and 920 features respectively.

Table II. COMPARISON OF EXECUTION TIMES FOR HOLOGRAM AND SEMEX.

	PROGRAM FUNCTION	# 1	# 2	# 3	# 4
H1	THRESHIT	15	15	15	15
S1	CLIP (Note 1)	7.5	7.5	7.5	7.5
	SPEEDUP (H1/S1)	2.0	2.0	2.0	2.0
H2	FEAT_ID	49	66	96	185
S2	TAG (Note 2)	12.9	14.4	18.1	35.1
	SPEEDUP (H2/S2)	3.8	4.6	5.3	5.3
H3	SIZEIT	52	125	343	620
S3	SIZE (Note 3)	7.9	8.2	14.6	25.4
	SPEEDUP (H3/S3)	6.6	15.2	23.5	24.4
H4	HOLOGRAM	166	256	504	870
S4	SEMEX (Note 4)	50	66	78	108
	SPEEDUP (H4/S4)	3.3	3.9	6.5	8.1

NOTES:

1. The times for steps H1 and S1 do not include times to input filenames (approximately 10 s each). The H1 times are based on one iteration only. In practice, 3 to 5 iterations may be required before arriving at a satisfactory threshold. For S1, the thresholds are determined automatically by *autoclip()*. These same thresholds are used in *THRESHIT* so that the same binary images are processed by both programs.

2. The times for steps H2 do not include times to input filenames. For S2, no filename entry is required as the frame memory is used.

3. The times for steps H3 do not include times to input filenames and answering prompts (approximately 25 s). For S3, no user intervention is required.

4. The times for H4 and S4 includes all the user input and setup times.

5. Four images (#1 to #4) were used with 48, 177, 495 and 920 features, respectively (see Figure 42).

been achieved. This is because the four nested loops used in both *THRESHIT* and *SIZEIT* (in the HOLOGRAM program) become highly inefficient. In SEMEX, this inefficiency is prevented by prematurely aborting the loops whenever the end of a feature is detected.

Another observation was that, for the same image and threshold, the modules within the HOLOGRAM program produced different feature counts. For example, with image #2, *THRESHIT* counted 177 features (same as *TAG* and *SIZE*) but *SIZEIT* counted only 175 features. Only image #1 produced consistent results for HOLOGRAM, whereas SEMEX produced consistent counts for all the four images. This suggests that one or more of the HOLOGRAM modules may not have been correctly coded.

C. CORRELATION WITH MALVERN MASTERSIZER

From one of the recent test firings using 4.69% aluminum solid-propellant, two sets of SEM images were extracted and analyzed. The results were then compared with that from the Malvern MasterSizer. The two sets of results are plotted in Figure 43 and Figure 44.

The results show that as the particle counts increase, the distributions produces a better correlation with the Malvern. Hockgraver [Ref. 6] showed that about 1,000 particles were required to produce a steady-state distribution for hologram images. From this particular set of images, 1,062 particles appeared to be insufficient. Unfortunately, there were no more SEM images from the same burn to extract. Hence, it is not possible to verify the minimum sample size requirement at this time.

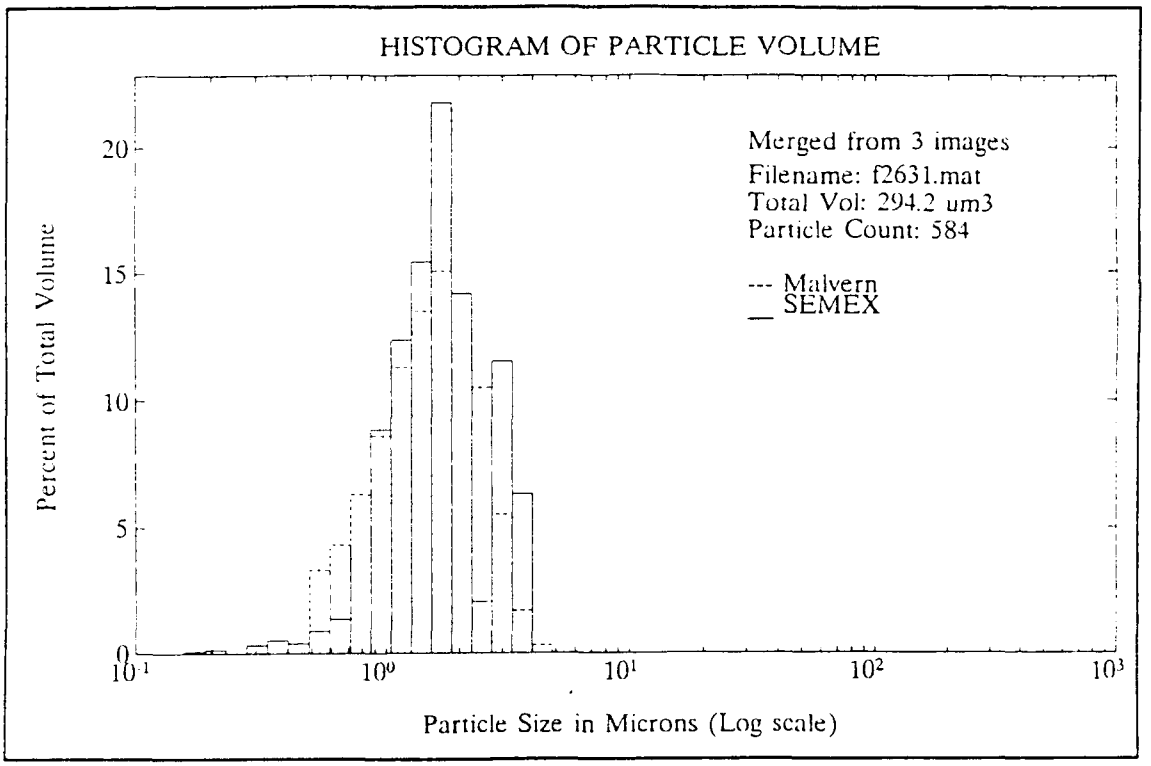

Figure 43. MATLAB plot showing the histogram data from SEMEX and Malvern MasterSizer. The SEMEX data was obtained from three images with a total of 584 particles.

It is suspected that an exact value cannot be determined even if sufficient SEM images were available. This is because the particle distribution varies considerably with the location of the probe tip (in both the radial and longitudinal directions), the portion of the filter paper from which the SEM images were taken, and the propellant characteristics. There is also a high possibility of debris and other contaminants being collected on the filter paper and this could skew the resulting particle distribution.

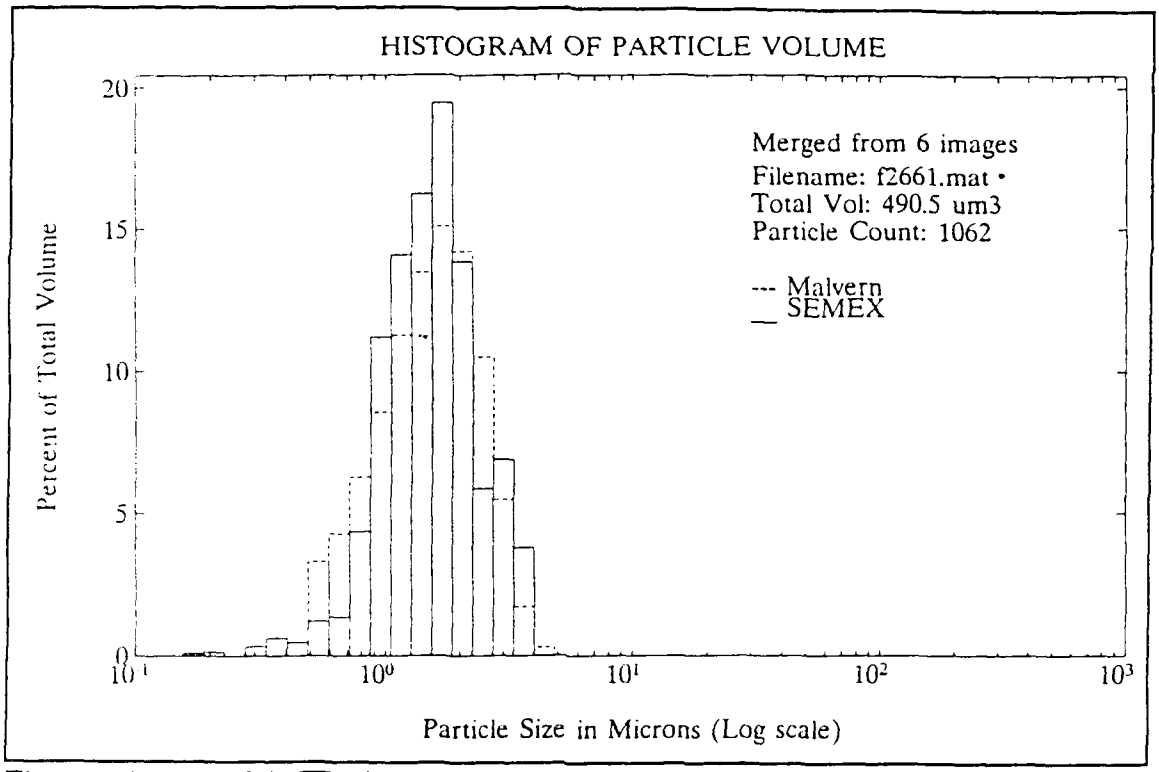

Figure 44. MATLAB plot showing histogram data from SEMEX and Malvern MasterSizer. The SEMEX data was obtained from six images with a total of 1062 particles.

VII. CONCLUSIONS AND RECOMMENDATIONS

This chapter lists the limitations of the present SEM extraction system and makes recommendations on how to further improve its performance. The recommendations are divided into three sections: hardware, software and methodology, respectively.

A. HARDWARE LIMITATIONS AND RECOMMENDATIONS

1. Light Table

The light table setup can be improved by using a more uniformly distributed light. The use of a fluorescent ring light with the camera lens placed though the center will help distribute the light uniformly on the photographic image. Other external illumination (from windows and ceiling) should be excluded. Use of diffusers would further reduce any uneven distribution of the illumination. A rigid photograph mount or stage would also be helpful for accurate alignment of the images. This would allow for the use of image subtraction techniques whereby an image of the blank background is first taken. This image is then subtracted from the actual SEM images to remove any dirt or artifacts present in both images.

2. Video Monitor

The present video monitor is unable to display the full image frame. Consequently, any edge artifacts cannot be seen. The use of a video monitor that

has adjustable vertical and horizontal static convergence controls (V-SIZE and H-SIZE) would enable all the frame borders to be seen.

3. Video Camera

The use of a Charge-Coupled Device (CCD) camera with square detector elements would overcome the problem of the pixel aspect ratio not being square and also eliminate the errors due to the scanning of the vidicon camera. In order to improve the accuracy of the extraction, higher resolution CCD cameras (typically 1024 x 1024 pixels) would be required. Unfortunately, these cameras are not RS-170 compatible and cannot be used with the existing frame grabber. The contrast transfer function (CTF) of the camera and lens system may have to be further investigated to determine its contribution to the slew rate errors.

4. Direct Acquisition of SEM Images

Presently, the Hitachi S450 SEM has a high persistence video display for the operator and a high resolution line scan for exposing the photograph. The problem of providing uniform illumination during acquisition of the SEM photographic images may be overcome by acquiring images directly off the SEM video display. However, the video display resolution is significantly poorer than the line scan. Hence, it is not advisable to record SEM images directly off the former.

A better arrangement would be to tap out the signal direct from the SEM line scan. The line scan takes approximately five seconds to cover the whole frame area. To use this signal would require building an interface with the correct video impedance matching and timing synchronization. The feasibility of this approach

would depend on the availability of technical information on the SEM. However, by mounting the video camera in place of the Polaroid camera, it may be possible to integrate all the frames (approximately 300, based on 5 s at 30 frames/second) to form a complete image.

An ideal solution would be to use a SEM that has a RS-170 video output and a programmable stage. In this approach, programmed instructions can be inserted to make the SEM stage scan all the areas on the sample without overlap. The images captured can be processed in real time if a faster processor is available. This will reduce the sampling errors due to small sample population.

5. Sun Workstation

Various optimization techniques have already been employed to enhance the speed of the system. This has resulted in an order of magnitude increase in speed over that of the HOLOGRAM program. Further attempts to increase the speed would require disproportionate amounts of effort (Amdahl's Law). Presently, the processing speed is limited by the IBM AT/386 and the frame grabber memory. In addition, the resolution of the current setup allows reliable sizing down to only 1/8 μm (with some degradation in accuracy). This resolution is limited by the video camera and the frame grabber.

By upgrading to a Sun SparcStation 1, it is expected that another order of magnitude increase in speed would be possible with a two-fold increase in resolution. However, the frame grabber, video copy processor and monitor would also have to be replaced. The Sun SparcStation 1 has a 32-bit SPARC CPU running at 1.25 MIPS

coupled with a floating-point unit for faster floating-point computations. The higher integer and floating-point performance over a PC would auger well for processing the larger image arrays. It has a 104 MB internal SCSI hard disk and a 3.5" 1.44 MB floppy disk drive for easy data transfer and DOS compatibility. The latter would allow the SEMEX program to be ported over. A higher-resolution frame grabber with function calls compatible with the PCVISION*plus* should be used to minimize software portability problems. The monitor resolution is 1152 (h) by 900 (v) pixels with a pixel aspect ratio of 1:1. The monitor has horizontal and vertical static convergence controls to adjust the frame size.

B. SOFTWARE ENHANCEMENTS

1. Automatic Camera Input Adjustment

Automated setup procedure for adjusting the camera inputs can be done by generating a histogram of the distribution of pixel gray scales. An image with good contrast would have a good spread of gray scales ranging from peak black to peak white. An algorithm could be devised that would analyze the gray scale distribution and would adjust the camera offset automatically to maximize image contrast. This would eliminate the subjective determination of the camera input gain and offset. The setup time could also be shortened, as there would be less user intervention required.

2. Automatic Threshold Algorithm

Currently, the background is sampled at fixed points which have been empirically-determined, based on the existing lighting arrangement. Changing the lighting could affect the performance of the automatic threshold algorithm. Consequently, the regions may not be optimal and may need to be changed. With faster processors available, the whole frame area can be scanned to determine the background threshold. This would result in a more accurate background threshold assessment.

3. Improved Image Processing Algorithms

Improved image processing, such as independent sizing of overlapping particles, filling in of hollow areas, and filtering off of irregularly-shaped features that fail certain roundness and sphericity tests, could be added. Currently, joined features are treated as one large feature with an irregular perimeter. However, if the curvature of each of the overlapped feature could be found, the outline of each of the overlapped features could be determined. This requires a significant amount of computation and, hence, was not implemented in SEMEX.

Up to this point, the only sphericity test carried out is by calculating the area of an equivalent ellipse using the x-chord and y-chord lengths. This could be improved by measuring other parameters, such as the perimeter or radius of curvature. In some cases, particularly when large particles are involved, it was noted that the centers of the particles were not filled. This would result in smaller measured areas (AREA_M) and could introduce false features within the hollow of

the particle (see Figure 45). *TAG* may have to be modified to test for the existence of unfilled areas and to fill these up automatically.

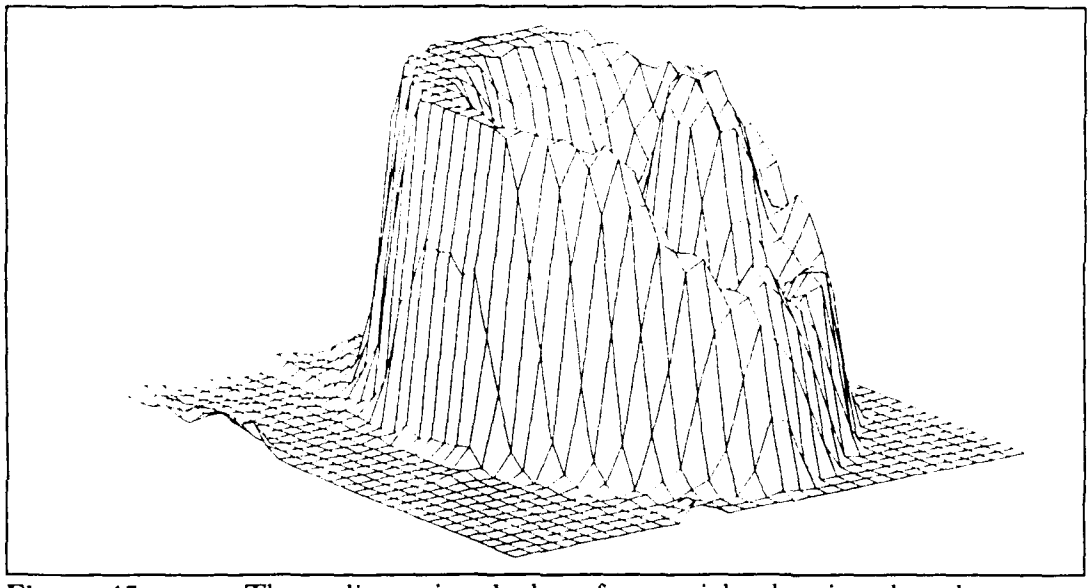

Figure 45. Three-dimensional plot of a particle showing that the center region is not a plateau. A smaller particle can be seen rising out of the center. This would give rise to two particle counts.

4. Frame Border

The addition of an area of interest may be justified if particles are likely to be small. This would then give rise to a border region whereby particles cutting the border would be sized and counted as a fraction of a particle, depending on the proportion inside the area of interest. Particles outside the area of interest are not sized. This will eliminate sizing errors due to incomplete particles. An alternative scheme would be to simply exclude all particles touching the edge of the frame.

C. IMPROVEMENTS IN METHODOLOGY

1. Photographing SEM Images

As far as possible, images should be taken with the same magnification and brightness and contrast settings. This would enable the acquisition stage to proceed faster. *SETUP* would have to be called only once and all the parameters would then be left unchanged for the whole set of images.

From the five sets of images processed, it was found that up to four exposures could be made on one photograph without significant degradation in the results. This saves time in the photographing process and also in the extraction process. The only limitation is where overlapping of particles begins to occur, due to the multiple exposures. This will result in larger particles being sized.

In chapter VI, more than 500 particles were used in the histogram plots. This was found to be insufficient. Hence, there should be sufficient SEM images to extract so that a steady-state distribution can be obtained.

2. Running SEMEX

Although SEMEX allows the user to customize the sequence of the processing, there is a preferred sequence to extracting SEM images, namely: acquire, clip, tag and size. This was shown in Figure 5 and the aim is to reduce disk reads and writes which takes between 5 and 8 seconds per image. An image which has been acquired and saved has its pixel values stored in the frame memory. Hence, CLIP can be initiated without recalling the image from disk. Similarly, after each clip and tag operation, the frame memory can be used by tag and size, respectively.

In this way, there is no need to retrieve an image from disk. Another advantage is that intermediate disk files need not be maintained (i.e., *.im1*, *.im2* and *.im3* files).

D. CONCLUSIONS

The IBM AT/386 running SEMEX has been found to speed up the extraction of particle sizes from SEM images. The results show that for the three stages of clipping, tagging and sizing, a speedup of nearly an order of magnitude was possible over that of the HOLOGRAM program. The speedup and throughput would be considerably more if all the stages of acquisition, clipping, tagging, sizing and analysis were compared. The errors obtained for small particle sizes on the order of half a micron or less are considerable. To reduce these errors, the magnification of the SEM images would have to be increased or a higher-resolution acquisition system would be required. The simplest approach would be to increase the magnification. This would require more photographic images and consequently more operator time and manpower cost. A higher-resolution system would entail an increase in computational cost on the order of the square of the frame length in pixels. This would strain the IBM AT/386 but would be comfortably handled by a Sun SparcStation. The capital outlay could be justified by the gain in system resolution and the savings in manpower cost.

APPENDIX A. NOTES TO THE PROGRAMMER AND THE USER

This appendix contains a description of the files and libraries used by SEMEX. It describes the process of compiling the source codes into an executable file using the Microsoft *MAKE* facility. The corresponding *makefile* (MAKESEM) is also listed. In addition, pointers to programmers and users are provided to help them understand and maximize the potential for this program.

A. SEMEX PROGRAM FILES

Figure 46 lists the files required to make up the SEMEX executable file. The files with the *.C* extension are the C source files. These are ASCII text files and can be edited with any ASCII text editor. The full listings of the source files are found in Appendix B. The GLOBAL.H file contains all the prototype definitions required by the C language for SEMEX; global variables and constants are also defined here. A full listing of GLOBAL.H is also found in Appendix B.

When compiled, each source file produces a corresponding *.OBJ* object file. The object files contain the machine language instructions required to perform the commands contained in the source codes. The process of linking resolves any external function calls or variables and binds the object codes and the libraries together to produce an executable file. After linking, the *SEMEX.EXE* file is created. This process of compiling and linking can be automated by using the *MAKE* facility provided by Microsoft. The *MAKE* command requires a *makefile* which

```
 Volume in drive F is LEE
 Directory of  F:\SEMEX

GLOBAL   H        4082   2-28-91  10:10a
ACQUIRE  C       10809   2-27-91   2:49p
CLIP     C        9063   2-27-91   3:04p
SEMIO    C        8967   2-27-91   3:06p
TAG      C       13987   2-28-91  11:00a
ANALYZE  C       20055   2-28-91   1:45p
SIZE     C       18975   2-28-91  11:03a
SETUP    C       26789   2-28-91   1:59p
SEMEX    C       10687   3-04-91   5:53p
TAG      OBJ      4809   2-28-91   1:48p
SEMIO    OBJ      3360   2-27-91   3:27p
SIZE     OBJ      9204   2-28-91   1:47p
ACQUIRE  OBJ      4348   2-27-91   3:25p
ANALYZE  OBJ      7965   2-28-91   1:45p
SETUP    OBJ     11023   2-28-91   2:00p
CLIP     OBJ      2941   2-27-91   3:26p
SEMEX    OBJ      3700   3-04-91   5:53p
SEMEX    EXE    127481   3-04-91   5:54p
IMGCVT   EXE     57405   1-22-91   4:03p
```

Figure 46. Listing of SEMEX C source files, object files and executable files using the DOS *DIR* command.

contains the file dependencies and the commands to be executed. The MAKESEM *makefile* used in developing the current version of SEMEX is shown in Figure 47. MAKE will determine the validity of the object files based on the dates last modified. If the source file has a date later than that of the corresponding object file, *MAKE* will recompile the source file to generate a new object file. If the object file is current, no action is taken. In this way, only files that have been modified will be compiled. This saves compilation time. Text between the number sign '#' till the end of the line are treated as comments and are ignored by *MAKE*.

The compile switches currently set are:

/AL This compiles the source file using the large memory model. Far pointers are allocated. By default, maximum optimization is used.

```
# Make file for semex.c and all the dependencies
#
# To execute type: make makesem
#
# Options for optimizing and no Codeview
OPT = /Ze
LOPT = /E /ST:8192
# Options for no-optimization and Codeview
#OPT = /Zi /Od
#LOPT = /CO /ST:8192 /F /PAC

# Library paths - LARGE ITEXPCplus library and LARGE Microsoft C library
LIB1 = \pcplus\itex\itexpcml
LIB2 = \msc\lib\llibce

acquire.obj: acquire.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 acquire.c

analyze.obj: analyze.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 analyze.c

clip.obj: clip.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 clip.c

semex.obj: semex.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 semex.c

semio.obj: semio.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 semio.c

setup.obj: setup.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 setup.c

size.obj: size.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 size.c

tag.obj: tag.c # makesem
        cl /c /AL $(OPT) /G2 /Fs /Sl 100 tag.c

semex.exe: makesem semex.obj semio.obj size.obj setup.obj tag.obj clip.obj
            acquire.obj analyze.obj
        link /NOD $(LOPT) semex semio size setup tag clip acquire analyze,,,
                                lwin $(LIB1) $(LIB2)
        del *.lst
```

Figure 47. *MAKEFILE* used in creating SEMEX.

/G2 This compiles using the 80286 code instead of the 8088 code.

/Ze This enables extensions to ANSI C and offers additional features beyond that provided by ANSI C. One particular feature required by SEMEX is the use of casts to produce lvalues (left-hand values).

/Sl 100 This sets the width of the listing file to 100 characters. If a compilation error is detected, MAKE will abort leaving a listing file which the user can refer to. If no errors are detected after linking, all the listing files are deleted.

The variables $(OPT) and $(LOPT) in MAKESEM are used for macro substitutions and will be replaced by their equivalent right-hand expressions specified at the beginning of the *makefile*. Two sets of equates are provided; one is for enable CodeView debugging while the other optimizes for speed. The former is currently commented out.

After successful compilation, the linking process will be initiated. As long as one object file has been modified, the *LINK* command will be invoked. This causes the object files and the libraries to be linked together. The libraries must be in the correct sub-directories as indicated by their paths. If the link is successful, all the listing files will be deleted. The libraries used are

LWIN.LIB	Large Memory Model Library for WINDOW BOSS. This currently resides in f:\SEMEX.
ITEXPCML.LIB	Large Memory Model Library for ITEX PC*plus*.
LLIBCE.LIB	Large Memory Model Library for Microsoft C.

B. SEMEX OUTPUT FILES

SEMEX may generate one or more output files. Some of these files are used by other modules in SEMEX, while others are files which the user can subsequently access. Figure 48 shows the files that can be generated by SEMEX and what they represent. SEMEX automatically appends the correct extension. The extension provided should not be changed, as this may confuse SEMEX. The filename can be entered by simply typing in the characters. The [Ins] key can be used to insert characters between existing characters and the [←] and [→] arrows keys can be used to position the cursor.

The image files are all stored in ITEX PC*plus* compressed file format. This generally achieves a storage efficiency of 1.8. However, for highly textured images, the compressed file may actually take up more disk space. With the exception of the *.mat* and *.met* files, the rest of the files are ASCII text files. It should be noted that non-ASCII files should not be read by a text editor as the latter may try to format the files by inserting special codes. This would destroy the integrity of the files render them unusable.

C. WORKING FROM DIFFERENT DIRECTORIES

Generally, subdirectories may be used to contain image files from a particular burn. In this case, SEMEX should be run from that particular subdirectory from which the images are to be processed. To do this, simply type the following commands. For example, if the images are contained in g:\SEM\f14, type

g: [Enter]

FILE EXT	DESCRIPTION AND USAGE
.ses	Sessions file used to record all the activities for the day. The filename is based on a 3-letter month code and a 2-digit date code (e.g., Mar28.ses).
.img	Image file generated by *ACQUIRE* if the save option is exercised. The image is generally cropped and complemented.
.im1	Image file generated by *CLIP* if the save option is exercised. The image has been clipped and can be read by the *TAG* module.
.im2	Image file generated by *TAG* if the save option is exercised. The image has been tagged and can be read by the *SIZE* module.
.im3	Image file generated by *SIZE* if the save image option is exercised. The image should be identical to *.im1*. This is provided merely for test and verification purposes.
.dat	This is the data file generated by *SIZE* if the save data option is exercised. The data consists of the fid, the calculated area (AREA_C), the measured area (AREA_M), the x-chord and the y-chord, for every feature that meets the size specifications (i.e., not rejected).
.his	This is the histogram data file generated by *ANALYZE*. It contains 38 rows of data. Each row represents one bin and contains the upper and lower bin limits, the volume in μm^3, the percentage of total volume, the feature count and the percentage of total features counted.
.mat	This is the MATLAB data file generated by *ANALYZE* and contains information similar to the histogram data file except that it is in a MATLAB format. This is the file which will be asked for by SEM.M.
.met	This is the MATLAB graphics output file which is generated when the MATLAB *meta* command is invoked. The *.met* files can be plotted by typing GRAPH when in DOS.

Figure 48. List of output files generated by SEMEX.

cd \SEM\f14 [Enter]

Then, to execute SEMEX from f:\SEMEX, type

f:\SEMEX\SEMEX [Enter]

To run MATLAB, first make sure that the MATLAB path is set. For example, if MATLAB resides on drive f: in a directory called \MATLAB, then the following line must be added to the CONFIG.SYS file

MATLABPATH=f:\MATLAB

In order for the path to be set, the IBM system would have to be reset by simultaneously pressing [Ctrl], [Alt] and [Del]. After the system has booted up, type MATLAB at the DOS prompt. The histogram plotting function is invoked by typing

sem [Enter]

when inside MATLAB. The function will request for a histogram filename. Type in the filename and the .his extension. It will then ask whether to input the Malvern data. If the user chooses to, the function will display each bin limit and request for the corresponding Malvern data. After the bins are filled, MATLAB will plot out the histogram. The MATLAB meta function can be invoked to save the plot which can then be printed out using the GRAPH.BAT command in DOS.

D. SPECIAL KEYS TO NOTE

Besides the keys mentioned above, the following keys are important and useful to note.

[Esc] This key is to terminate SEMEX when in the main menu. It can also be used to abort any unwanted command. For example, *SIZE* usually proceeds automatically after *TAG* finishes (unless disabled during

SETUP). However, if the user finds that the tagging operation has produced wrong results (possibly because the size parameters have been wrongly set), then he or she could press the [Esc] key when any prompt appears. This will terminate *TAG* and return the user to the main menu.

[F1] This function key is used to obtain help information. The help is context-sensitive, where available.

Mouse support is not available at this time. However, the programmer can refer to the WINDOW BOSS manual on how to incorporate functions supporting the mouse.

APPENDIX B. PROGRAM LISTINGS

This appendix contains all the source code listings for SEMEX and its modules. The listings are documented with comments at the start of each function and at the end of each line. These are marked with a slash and an asterisk in the following way, /* Comment */. Also included is the MATLAB script file SEM.M. The sequence of the listings are as follows:

GLOBAL.H

SEMEX.C

SETUP.C

ACQUIRE.C

CLIP.C

TAG.C

SIZE.C

ANALYZE.C

SEM.M

```
/*********************************************************************
 * GLOBAL.H    Include files for use with SEMEX programs
 *********************************************************************
 */
#include <math.h>
#include "itexpfg.h"            /* ITEX PCplus function definitions */
#include "stdtyp.h"             /* ditto */
#include "window.h"             /* WINDOW BOSS function definitions */

/*  Program constants for the SEM Extraction (SEMEX) program
 *  used in conjunction with the PCVISIONplus Frame grabber and
 *  ITEXPCplus library functions */

/* PCVISIONplus Board Settings   */
#define  MEMBASE    0xD0000L    /* base memory start address */
#define  REGBASE    0x300       /* base register start address */
#define  MEMORY     DUAL        /* memory type */

/* Frame Dimensions */
#define XSIZE     512           /* Number of pixels in X direction */
#define YSIZE     512           /* Number of pixels in Y direction */
#define DEPTH     8             /* Number of bits per pixel */

/* AOI (area of interest) settings  */
#define IXS       0             /* Initial X Starting Point */
#define IYS       0             /* Initial Y Starting Point */
#define NROW      480           /* Total Number of rows in image */
#define NCOL      512           /* Total Number of columns in image */
#define LASTROW   479           /* Last row in image */
#define LASTCOL   511           /* Last column in image */

/*  Threshold Limits  */
#define  LOWEST          0      /* Equates to Black for lowcut value */
#define  HIGHEST         255    /* Equates to White for highcut value */
#define  BLACK_LEVEL     0      /* Indicates a feature */
#define  WHITE_LEVEL     255    /* Represents background */
#define  HIGH            254    /* Highest Tag value */
#define  LOW             1      /* Lowest Tag Value */
#define  GRAY            128    /* Middle Gray level */

/* Miscellaneous Limits AND Defaults */
#define MAXFLEN         20      /* Max filename length */
#define MAXLEN          200     /* Max comment length allowed by ITEX */
#define MAXCLEN         50      /* Comment length */

/* Define enter key */
#define ENTER           0x0d    /* [ENTER] key signature */
```

```
/*
 *  Global Variables Declarations
 */
extern char filename[MAXFLEN];   /* image filename */
extern char session[MAXFLEN];    /* session filename */
extern char comline[MAXLEN];     /* entire comment line */
extern char comline1[MAXCLEN];   /* comment line 1 */
extern char comline2[MAXCLEN];   /* comment line 2 */
extern FILE *fp;                 /* session file pointer */
extern float ASPECT_RATIO;       /* X to Y aspect ratio of a pixel */
extern long TOTAL;               /* Total feature counter */
extern int OVERSIZE;             /* Default too large feature */
extern int UNDERSIZE;            /* Default too small feature */
extern int GAIN_LVL;             /* Initial Gain level */
extern int OFFSET_LVL;           /* Initial Offset level */
extern int LT_MARGIN;            /* Initial Left Margin */
extern int RT_MARGIN;            /* Initial Right Margin */
extern int DF_GAIN ;             /* Default gain flag */
extern int DF_OFFSET;            /* Default Offset flag */
extern int DF_LM;                /* Default left margin flag */
extern int DF_RM;                /* Default right margin flag */
extern float VSCALE;             /* Initial vertical scale factor */
extern int DF_VSCALE;            /* Default vertical scale factor flag */
extern int DF_INVERT;            /* Auto Complement flag */
extern int LOAD_RAW;             /* Auto load RAW image flag */
extern int LOADCLIP;             /* Auto load Clipped image flag */
extern int DF_SIZE;              /* Default size limits flag */
extern int LOADTAG;              /* Ask to load tagged image flag */
extern int DO_SEQ;               /* Flag to sequence through whole process */
extern int HELP_LVL;             /* Help Level */

/*
 * External function prototypes found in SEMIO.C
 */
extern int getim(WINDOWPTR w,int n);   /* read image function */
extern int putim(WINDOWPTR w,int n);   /* save image function */
extern void chgext(char *fd, char *fs, char *ext);
                                       /* change extension function */


/* End of file GLOBAL.H */
```

```
/* ************************************************************************
 * FILENAME : SEMEX.C
 * LAST MODIFIED: 12 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE : Scanning Electron Microsope Extraction Program (SEMEX)
 *           This program uses extensive windowing and calls the
 *           following functions:
 *              acquire()  - acquire images from SEM photographs using
 *                           vidicon camera and PCVISIONplus framegrabber
 *              clipmain() - clips (threshold) the image
 *              tagmain()  - tags the features on the image
 *              sizemain() - sizes the features
 *              analyze()  - analyze the results by building histogram
 *              setup()    - set up equipment and change the level of
 *                           user control
 * ************************************************************************
 */

#include "global.h"                 /* global defines */
#include <time.h>                   /* timing prototypes */
#include <graph.h>                  /* graphics prototypes */
#include <dos.h>                    /* dos function prototypes */
#include <string.h>                 /* string handling prototypes */

/*
 *  Global Variables Defines
 */
char filename[MAXFLEN];             /* image filename */
char session[MAXFLEN];              /* session filename */
char comline[MAXLEN];               /* entire comment line */
char comline1[MAXCLEN];             /* comment line 1 */
char comline2[MAXCLEN];             /* comment line 2 */
FILE *fp;                           /* session file pointer */
float ASPECT_RATIO = 1.200;         /* X to Y pixel aspect */
long TOTAL;                         /* keeps track of total feature count */

/* The following can be redefined during run time by using setup() */
int GAIN_LVL      = 0    ;        /* Initial Gain - set to highest */
int OFFSET_LVL    = 60   ;        /* Initial Offset - set to midpoint */
int DF_GAIN       = FALSE;        /* Don't use default gain */
int DF_OFFSET     = FALSE;        /* Don't use default Offset */
int LT_MARGIN     = 0    ;        /* Initial Left Margin - leftmost */
int RT_MARGIN     = 512  ;        /* Initial Right Margin */
int DF_LM         = TRUE ;        /* Use default left margin = 0 */
int DF_RM         = FALSE;        /* Don't use default right margin */
int OVERSIZE      = 100  ;        /* Default too large feature */
int UNDERSIZE     = 1    ;        /* Default too small feature */
int DF_SIZE       = TRUE ;        /* Use default size limits */
float VSCALE      = 1.0  ;        /* Default Vertical scale factor */
int DF_VSCALE     = TRUE ;        /* Use default scale */
int DO_SEQ        = TRUE ;        /* Auto sequence through whole process */
int DF_INVERT     = TRUE ;        /* Complement Automatically */
int LOAD_RAW      = FALSE;        /* Don't load RAW image automatically */
int LOADCLIP      = FALSE;        /* Don't load Clipped image automatically */
int LOADTAG       = FALSE;        /* Don't load Tagged image automatically */
int HELP_LVL      = TRUE ;        /* Enable help screens */
unsigned blue     = BLUE ;        /* remap for mono */
```

```
main(void)
{
    /* Prototype declarations */
    extern int setup(void);          /* setup function */
    extern int acquire(void);        /* Acquire images from camera */
    extern int clipmain(void);       /* Clip Image */
    extern int tagmain(void);        /* Feature Tagging */
    extern int sizemain(void);       /* Feature Sizing */
    extern int analyze(void);        /* Analyse Feature data */
    extern clock_t clock(void);      /* Returns Number of clock ticks */
    extern void fginit(void);        /* Frame Grabber Initialization */
    extern void session_name(void); /* Use date/time as session name */

    WINDOWPTR wn;                    /* One Window */
    int i;                           /* scratch integer */
    int watrib,batrib;               /* scratch atributes */
    int rv = 0;                      /* for menu choice */
    int rerr = 0;                    /* return value from functions */
    int row,col;                     /* for positioning windows */
    clock_t start;                   /* Time variable */

    static struct pmenu smenu = {            /* define main menu */
      0, FALSE, 0,                           /* page 0, window open, indx */
      3,  8, {                               /* accept menu field 3 thru 8 */
      2, 11,          "S E M E X", 0,        /* field 0 - info */
      3,  4,  "SEM Extraction Program", 0,   /* field 1 - info */
      4,  3, "Naval Postgraduate School", 0, /* field 2 - info */
      6,  6, "1. Setup SEMEX      ", 1,      /* field 3 - choice 1 */
      7,  6, "2. Acquire Image    ", 2,      /* field 4 - choice 2 */
      8,  6, "3. Clip Image       ", 3,      /* field 5 - choice 3 */
      9,  6, "4. Tag Features     ", 4,      /* field 6 - choice 4 */
     10,  6, "5. Size Features    ", 5,      /* field 7 - choice 5 */
     11,  6, "6. Analyze Features", 6,       /* field 8 - choice 6 */
     13,  3, "Use cursor keys to select", 0, /* field 9 - info */
     14,  3, "Press [Enter] to execute ", 0, /* field 10 - info */
     15,  3, "Press [Esc] to quit      ", 0, /* field 11 - info */
     99, 99, "",99 }                         /* menu terminator */
    };

    start = clock();                    /* start timing */
    if(wns_mtflg == 7) blue = BLACK;    /* remap if mono */

    printf("Initializing.  Please wait...");
    fginit();                           /* set up Frame Grabber */
    strcpy(filename,"              .");  /* blank out filename */
    session_name();                     /* use date/time for session */
    watrib = v_setatr(WHITE,BLACK,0,BOLD);/* window attribute */
    for(i=0; i<25; i++) {               /* build the back drop */
        v_locate(0,i,0);                /* position cursor */
        v_wca(0, 0xb0, watrib, 80);     /* the fast way */
    }
    v_hidec();                          /* hide the cursor */
    wn_init();                          /* save entry screen */
```

```
    /*
     * Popup Menu
     */
    do {
        watrib = WHITE<<4|BLACK;            /* window colors */
        batrib = blue<<4|WHITE;             /* border colors */
        if( (!DO_SEQ) || (rv < 3) || rerr ) /* popup menu to get user choice */
            rv = wn_popup(0,3,23,31,17,watrib,batrib,&smenu,FALSE);
        else {                              /* auto sequence */
            rv++;                           /* get next menu list */
            if (rv > 5) rv = 0;             /* end of menu list */
        }
        crystal();                          /* set to internal sync */
        switch (rv) {                       /* do user command */
            case 1:                         /* setup configuration */
                rerr = setup();
                break;
            case 2:
                rerr = acquire();           /* Acquire Image */
                break;
            case 3:
                rerr = clipmain();          /* Clip Image */
                break;
            case 4:
                rerr = tagmain();           /* Feature Tagging */
                break;
            case 5:
                rerr = sizemain();          /* Feature Sizing */
                break;
            case 6:                         /* Analyse Feature data */
                rerr = analyze();
                break;
            case 99:                        /* error or ESC key */
            default:
                rerr = 1;
                break;
        } /* end switch */
    } while(rv !=99);
    wn_exit();                              /* restore entry screen */
    _clearscreen(_GCLEARSCREEN);            /* clear screen */
    printf("\n\n\n\n\tSession's Activities have been recorded in %s",session);
    printf("\n\n\tSEMEX was on for %.1f minutes.",
         (float) (clock()-start) / (float) (CLK_TCK * (clock_t) 60) );
    printf("\n\n\tDeveloped by ECE Dept, Naval Postgraduate School");
    printf("\n\n\tHave A nice day!\n\n\n");
    if(( fp = fopen(session,"a")) == NULL)
        printf("a\n\n\Unable to open session file %s\n\tSEMEX Aborted.", session);
    else {
        fprintf(fp,"\nSEMEX was on for %.1f minutes.",
                 (float) (clock()-start) / (float) (CLK_TCK * (clock_t) 60) );
        fclose(fp);
    } /* if-else */
    exit(0);                                /* Successful termination */
}
```

```
/*
 * Initial PCVISIONplus Frame Grabber setup (Refer to global.h for defines)
 */
void
fginit(void)
{
     sethdw(REGBASE,MEMBASE,MEMORY);     /* Set hardware definitions */
     setdim(XSIZE,YSIZE,DEPTH);          /* set frame dimensions */
     fgon();                             /* turn on frame grabber */
     initialize();                       /* set up registers and LUTs */
     setcamera(0);                       /* Select Camera */
     extsync();                          /* Use Camera video sync */
     waitvb();                           /* sync to vertical blanking */
     select_mem(MEM_A);                  /* Select frame A */
     display_mem(MEM_A);                 /* display frame A */
     sclear(GRAY);                       /* clear whole TV screen */
     setlut(INLUT,0);                    /* Use INPUT LUT, bank 0 */
     setlut(GRNLUT,0);                   /* Use GREEN OUTPUT LUT, bank 0 */
}

/* session_name() gets the current date and time and creates a session
 * filename
 */
void
session_name(void)
{
    struct dosdate_t date;               /* dos.h date structure */
    struct dostime_t time;               /* dos.h time structure */
    static char *month[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                               "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

    _dos_getdate (&date);                /* get current date */
    _dos_gettime (&time);                /* get current time */
    if (date.day < 10)                   /* single digit date, pad zero */
        sprintf(session,"%3s0%1d.ses          ",month[date.month - 1], date.day);
    else                                 /* double digit date */
        sprintf(session,"%3s%2d.ses           ",month[date.month - 1], date.day);
    if ( (fp = fopen(session,"a")) == NULL) {
        printf("a\n\n\Unable to open session file %s\nSEMEX Aborted.", session);
        exit;                            /* terminate */
    }
    else {
        if (time.minute < 10)            /* single digit minute, pad zero */
            fprintf(fp,"\nOpening Session on %2d %3s %4d at %2d:0%1d.",
                date.day, month[date.month - 1], date.year, time.hour, time.minute);
        else                             /* double digit minute */
            fprintf(fp,"\nOpening Session on %2d %3s %4d at %2d:%2d.",
                date.day, month[date.month - 1], date.year, time.hour, time.minute);
        fprintf(fp,"\n-----------------------------------------\n");
        fclose(fp);                      /* close session file */
    }
}


/* End of file SEMEX.C */
```

```
/* ************************************************************************
 * FILENAME: SETUP.C
 * CALLED BY: semex() and calls clip()
 * LAST MODIFIED: 12 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE: Setup is used to set up the SEMEX program to configure it
 *          to suit the user's needs.  Various prompts can be turned on
 *          or off to stream-line the processing of SEMs.
 *          It changes the set of global variables and flags which the
 *          various functions in SEMEX make use of, or test for.
 *          Latest modification incorporates check_equipment which
 *          facilitates the setup of the light table and camera.
 *          measure_line() has also been added to determine the
 *          magnification of the image.
 * ************************************************************************
 */
#include "global.h"
#include <string.h>

setup()
{
    extern void bool2YN(int, char *);       /* prototype declaration */
    extern void check_equipment(void);      /* ditto */
    WINDOWPTR wn;                           /* one window pointer */
    WIFORM frm;                             /* form pointer */
    int watrib, batrib;                     /* window, border and */
    unsigned fatrib;                        /* field attributes */
    static char *emsg1 = "Only range of 0 to 100 allowed.  Press any key";
    static char *emsg2 = "Only range 2 to 256 allowed.  Press any key";
    static char *emsg3 = "Only range 1 to 255 allowed.  Press any key";
    static char *emsg4 = "Only range 0 to 255 allowed. Press any key";
    static char *emsg5 = "Only range 256 to 512 allowed. Press any key";
    char b0[5], b1[3];                      /* string buffers */
    char b2[5], b3[3];                      /* ditto */
    char b4[5], b5[3];                      /* ditto */
    char b6[5], b7[3];                      /* ditto */
    char b8[9], b9[3];                      /* ditto */
    char b10[5], b11[5], b12[3];            /* ditto */
    char b13[6], b14[3];                    /* unused at the moment */
    char b15[3], b16[3], b17[3], b18[3];    /* Y/N input buffers */
    char b19[3], b20[3];                    /* ditto */

    Check_Equipment();                      /* check camera and lights */
    fatrib = (BLUE<<4) | WHITE | BOLD;      /* field color */
    watrib = v_setatr(WHITE,BLUE,0,0);      /* window color */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* border color */
    wn = wn_open(500,1,5,60,19,watrib,batrib);
    if (!wn) {
        printf("\a\n Unable to open window.  Aborting...");
        exit(1);
    }
    wn_title(wn," SETUP DEFAULTS ");
    frm = wn_frmopn(22);                    /* open 21 + 1 fields */
    if (!frm) {
        printf("\a\n Unable to open form.  Aborting...");
        exit(1);
    }
    itoa(GAIN_LVL,b0,10);
    wn_gint(SET,frm, 0,wn, 2, 1,"GAIN LEVEL        :  ",fatrib,'_',
            &GAIN_LVL,3,0,100,b0,emsg1,emsg1);
    bool2YN(DF_GAIN,b1);
    wn_gbool(SET,frm, 1,wn, 2,30,"Use Default[Y,N] : ",fatrib,'_',
            &DF_GAIN,b1,NSTR,NSTR);
    itoa(OFFSET_LVL,b2,10);
    wn_gint(SET,frm, 2,wn, 3, 1,"OFFSET LEVEL      :  ",fatrib,'_',
           &OFFSET_LVL,3,0,100,b2,emsg1,emsg1);
    bool2YN(DF_OFFSET,b3);
    wn_gbool(SET,frm, 3,wn, 3,30,"Use Default[Y,N] : ",fatrib,'_',
            &DF_OFFSET,b3,NSTR,NSTR);
```

```
    itoa(LT_MARGIN,b4,10);
    wn_gint(SET,frm, 4,wn, 4, 1,"LEFT MARGIN       :  ",fatrib,'_',
            &LT_MARGIN,3,0,511,b4,emsg4,emsg4);
    bool2YN(DF_LM,b5);
    wn_gbool(SET,frm, 5,wn, 4,30,"Use Default[Y,N] : ",fatrib,'_',
             &DF_LM,b5,NSTR,NSTR);
    itoa(RT_MARGIN,b6,10);
    wn_gint(SET,frm, 6,wn, 5, 1,"RIGHT MARGIN      :  ",fatrib,'_',
            &RT_MARGIN,3,1,512,b6,emsg5,emsg5);
    bool2YN(DF_RM,b7);
    wn_gbool(SET,frm, 7,wn, 5,30,"Use Default[Y,N] : ",fatrib,'_',
             &DF_RM,b7,NSTR,NSTR);
    sprintf(b8,"%7.3f",VSCALE);
    wn_gfloat(SET,frm,8,wn, 6, 1,"Y-SCALE FACTOR : ",fatrib,'_',
              &VSCALE,7,3,0.0,999.0,b8,emsg1,emsg1);
    bool2YN(DF_VSCALE,b9);
    wn_gbool(SET,frm, 9,wn, 6,30,"Use Default[Y,N] : ",fatrib,'_',
             &DF_VSCALE,b9,NSTR,NSTR);
    itoa(OVERSIZE,b10,10);
    wn_gint(SET,frm,10,wn, 7, 1,"Max Feature Size  :  ",fatrib,'_',
            &OVERSIZE,3,2,256,b10,emsg2,emsg2);
    itoa(UNDERSIZE,b11,10);
    wn_gint(SET,frm,11,wn, 8, 1,"Min Feature Size  :  ",fatrib,'_',
            &UNDERSIZE,3,1,255,b11,emsg3,emsg3);
    bool2YN(DF_SIZE,b12);
    wn_gbool(SET,frm,12,wn, 8,30,"Use Defaults[Y,N]: ",fatrib,'_',
             &DF_SIZE,b12,NSTR,NSTR);
    bool2YN(HELP_LVL,b15);
    wn_gbool(SET,frm,13,wn,10, 1,"ALL: Enable HELP screens [Y,N]                    : ",
             fatrib,'_',&HELP_LVL,b15,NSTR,NSTR);
    bool2YN(DO_SEQ,b16);
    wn_gbool(SET,frm,14,wn,11, 1,"SEMEX: CLIP, TAG and SIZE without asking[Y,N]: ",
             fatrib,'_',&DO_SEQ,b16,NSTR,NSTR);
    bool2YN(DF_INVERT,b17);
    wn_gbool(SET,frm,15,wn,12, 1,"ACQUIRE: Complement Image without asking[Y,N]: ",
             fatrib,'_',&DF_INVERT,b17,NSTR,NSTR);
    bool2YN(LOAD_RAW,b18);
    wn_gbool(SET,frm,16,wn,13, 1,"CLIP: Load RAW Image without asking[Y,N]       : ",
             fatrib,'_',&LOAD_RAW,b18,NSTR,NSTR);
    bool2YN(LOADCLIP,b19);
    wn_gbool(SET,frm,17,wn,14, 1,"TAG: Load CLIPPED Image without asking[Y,N]    : ",
             fatrib,'_',&LOADCLIP,b19,NSTR,NSTR);
    bool2YN(LOADTAG,b20);
    wn_gbool(SET,frm,18,wn,15, 1,"SIZE: Load TAGGED Image without asking[Y,N]    : ",
             fatrib,'_',&LOADTAG,b20,NSTR,NSTR);
    wn_gtext(SET,frm,19,wn,16, 1,"Session Filename: ",fatrib,'_',18,session,NSTR,NSTR);
    wn_dtext(SET,frm,20,wn,18,1,"Press [Esc] to accept Existing defaults");

    if(!wn_frmget(frm)) {              /* read form */
        printf("\a\nMemory corrupted.  Aborting wn_frmget()");
        exit(1);
    }
    wn_frmcls(frm);                    /* close form */
    wn_close(wn);                      /* close window */
    return(0);
}

/*
 * Convert 1's to Y's and 0's to N's
 */
void
bool2YN(int n, char *s)
{
    if (n == TRUE)
       strcpy(s,"Y");
    if (n == FALSE)
       strcpy(s,"N");
}
```

```
/* ************************************************************************
 * FUNCTION NAME: check_equipment()
 * PURPOSE  : This function allows setting up of the light table and
 *            camera prior to actual acquisition of images.
 *            Afer acquiring a test image, it can be inverted and
 *            clipped to ascertain the uniformity of illumination.
 *            A graphic cursor facility is also provided to measure
 *            various features.
 * ************************************************************************
 */
void check_equipment(void)
{
    extern int measure_line(WINDOWPTR w);    /* function prototype */
    extern int clip(WINDOWPTR w, int t);     /* ditto */
    WINDOWPTR wn;                            /* window handle */
    int watrib, batrib;                      /* Window, border and */
    unsigned fatrib;                         /* field attributes */
    char c;                                  /* scratch for user response */
    int i;                                   /* scratch index */
    int done;                                /* scratch flag */
    int gain, offset;                        /* gain and offset setting */

    watrib = v_setatr(WHITE,BLUE,0,0);       /* set window attribute */
    batrib = v_setatr(RED,WHITE,0,BOLD);     /* set border attribute */
    wn = wn_open(500,8,13,50,14,watrib,batrib); /* open dialog window */
    if (!wn) {
       printf("\a\n\n Unable to open window");
       exit(1);
    }
    wn_title(wn," EQUIPMENT SETUP ");
    wn_printf(wn,"\n\n\tSet up camera and lights[Y] ?");
    v_kflush();                              /* empty keyboard buffer */
    c = getch();                             /* get user response */
    while ( (c != 'n') && (c != 'N') ) {     /* repeat until No */
        /* Turn on camera */
        wn_clr(wn);                          /* clear window */
        wn_title(wn," SET UP CAMERA AND LIGHTS ");
        wn_puts(wn,2,5,"Turning on GRAB Mode...");
        setcamera(0);                        /* connect camera 0 */
        extsync();                           /* external sync */
        grab(NO_WAIT);                       /* ensure grab mode ON */
        grab(NO_WAIT);
        /* Fine tuning the board's gain and offset */
        wn_puts(wn,1,5," GAIN AND OFFSET SETTING ");
        gain = GAIN_LVL;
        setgain(gain);                       /* initial gain setting */
        v_kflush();                          /* prevent spurious input */
        wn_puts(wn,7,1,"Use [+] and [-] keys to adjust");
        wn_puts(wn,8,1,"Press [ENTER] to continue");
        wn_puts(wn,2,1,"Gain (0 highest, 100 lowest): ");
        wn_printf(wn,"%3d",gain);
        while ( (c=getch()) != ENTER )
        {
            wn_locate(wn,2,31);
            if ( (c == '+') && (gain < 100) )
              gain = gain + 5;               /* increment in steps of 5 */
            if ( (c == '-') && (gain > 0) )
              gain = gain - 5;               /* decrement in steps of 5 */
            setgain(gain);                   /* change the camera gain */
            wn_printf(wn,"%3d",gain);
        }
        GAIN_LVL = gain;                     /* Update new gain level */
        DF_GAIN = TRUE;                      /* Use default gain */
        offset = OFFSET_LVL;
        setoffset(offset);
        wn_puts(wn,4,1,"Offset (0 darkest, 100 lightest): ");
        wn_printf(wn,"%3d",offset);
        while ( (c=getch()) != ENTER)
        {
```

```
            wn_locate(wn,4,35);
            if ( (c == '+') && (offset < 100) )
              offset = offset + 5;          /* increment in steps of 5 */
            if ( (c == '-') && (offset > 0) )
              offset = offset - 5;          /* decrement in steps of 5 */
            setoffset(offset);              /* change the camera offset */
            wn_printf(wn,"%3d",offset);
        }
        OFFSET_LVL = offset;                /* Update new offset */
        DF_OFFSET = TRUE;                   /* Use default offset */
        /*  Snap a frame (this will stop further acquisition) */
        wn_title(wn," ACQUIRE IMAGE - SNAP MODE ");
        done = FALSE;                       /* check whether image snapped */
        wn_clr(wn);
        v_kflush();                         /* prevent spurious input */
        wn_printf(wn,"\n\n\tPress [SPACEBAR] to snap an image");
        wn_printf(wn,"\n\t  repeat until satisfied");
        while ( (c = getch()) != ENTER)
        {
            wn_puts(wn,6,4,"Wait...");
            waitvb();                       /* Wait for vertical blanking */
            snap(WAIT);                     /* acquire a frame */
            wn_puts(wn,6,4,"Done!  ");
            wn_puts(wn,12,4,"When done, press [ENTER]");
            done = TRUE;                    /* flag that image taken */
        }
        if (!done) {
            waitvb();                       /* Wait for vertical blanking */
            snap(WAIT);                     /* turn grab off */
        }
        crystal();                          /* internal sync for stability */
        setcamera(1);                       /* disconnect camera 0 */
        c = 'Y';
        while ((c == 'y') || (c == 'Y') ) { /* repeat */
            wn_clr(wn);                     /* clear screen */
            if (measure_line(wn))           /* measure line lengths */
                return;                     /* error detected.  Abort */
            wn_printf(wn,"\n\n\tMeasure another feature [N]?");
            v_kflush();                     /* empty keyboard buffer */
            c = getch();
        }
        wn_clr(wn);
        /* Complement Image in frame memory */
        v_kflush();                         /* empty keyboard buffer */
        wn_printf(wn,"\n\n\tComplement Image [Y]?");
        c = getch();
        if ( (c!='N') && (c!='n') ) {
            DF_INVERT = TRUE;               /* Set auto-invert flag */
            wn_printf(wn,"\n\tComplementing Image...");
            complement(IXS,IYS,NCOL,NROW);
        }
        else
            DF_INVERT = FALSE;              /* Reset auto-invert flag */
        /* Threshold image */
        wn_printf(wn,"\n\n\tUpdating frame memory...");
        maplut(GRNLUT,0,IXS,IYS,NCOL,NROW); /* update frame memory */
        wn_clr(wn);                         /* clear window */
        threshold(GRNLUT,0,HIGHEST,HIGH);   /* Initial threshold */
        wn_printf(wn,"\n\tReduce to guage Lighting Uniformity");
        clip(wn,HIGH);                      /* threshold image */
        wn_clr(wn);                         /* clear window */
        wn_printf(wn,"\n\n\tAdjust Lighting and Try again[Y] ?");
        v_kflush();                         /* clear keyboard buffer */
        c = getch();                        /* get user response */
        linlut(GRNLUT,0);                   /* restore LUT */
    } /* end while */
    wn_close(wn);                           /* all done */
    return;                                 /* return */
}
```

```
/* ************************************************************************
 * FUNCTION NAME: measure_line()
 * CALLED BY: check_equipment()
 * LAST MODIFIED: 7 Mar 91 by  LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE:  This provides a set of functions for generating a graphics
 *           cursor (cross-hair), moving it around with the directional
 *           keys and drawing vertical and horizontal lines.  The
 *           directional keys function as follows:
 *              Up      Moves cross-hair up one pixel at a time
 *              Down    Moves cross-hair down one pixel at a time
 *              Left    Moves cross-hair left one pixel at a time
 *              Right   Moves cross-hair right one pixel at a time
 *              Home    Moves cross-hair left in steps
 *              End     Moves cross-hair right in steps
 *              PgUp    Moves cross-hair up in steps
 *              PgDn    Moves cross-hair down in steps
 *           Presently step size is 10 pixels
 *           It also calculates the magnification of a 5 micron line.
 ************************************************************************
 */
/* #include "global.h" */

#undef UARROW                           /* undefine to prevent conflict */
#undef DARROW                           /* ditto */
#undef LARROW                           /* ditto */
#undef RARROW                           /* ditto */
#undef PAGEUP                           /* ditto */
#undef PAGEDN                           /* ditto */
#undef HOME                             /* ditto */
#undef END                              /* ditto */

#define UARROW 0x48                     /* redefine key constants */
#define DARROW 0x50                     /* ditto */
#define LARROW 0x4b                     /* ditto */
#define RARROW 0x4d                     /* ditto */
#define PAGEUP 0x49                     /* ditto */
#define PAGEDN 0x51                     /* ditto */
#define HOME   0x47                     /* ditto */
#define END    0x4f
#define STEP   10                       /* step size for fast move */

static int xpixval[9], ypixval[9];      /* pixal value under cursor */
static int line[NROW];                  /* allocate memory for a line */

/*
 * measure a line marked by two graphic cursors
 */
measure_line(WINDOWPTR w)
{
    extern void put_cursor(int x,int y);    /* function prototype */
    extern void unput_cursor(int x,int y);                  /* ditto */
    extern void put_line(int x1,int y1,int x2,int y2);      /* ditto */
    extern void unput_line(int x1,int y1,int x2,int y2);    /* ditto */
    extern void chkkey(char c, int *x, int *y);             /* ditto */
    extern float calc_line(int x1,int y1,int x2,int y2);    /* ditto */
    char c;                             /* scratch for key pressed */
    int x = 470, y = 65;                /* initial cursor location */
    int x1, y1;                         /* store 1st point on line */
    float len;                          /* length of line */
    int color;                          /* pixel color */

    v_kflush();                         /* prevent spurious inputs */
    wn_printf(w,"\n\n\tLine Measure Feature");
    wn_printf(w,"\n\n\tPosition cursor at first point");
    wn_printf(w,"\n\tusing ARROW keys.  Press [ENTER] when done\n");
    put_cursor(x,y);                    /* put cursor at the center */
    wn_printf(w,"\r\tCoordinates:     X1: %3d   Y1: %3d", x,y);
    while ( (c = getch()) == '\0')      /* if cursor direction key */
```

```
    {
        c = getch();                        /* check direction */
        unput_cursor(x,y);                  /* remove old cursor */
        chkkey(c, &x, &y);
        put_cursor(x,y);                    /* put cursor back on */
        wn_printf(w,"\r\tCoordinates:     X1: %3d   Y1: %3d", x,y);
    } /* end while */
    x1 = x;                                 /* save 1st point */
    y1 = y;
    put_line(x1,y1,x,y);                    /* put out a line */
    /* get second point on the line */
    wn_printf(w,"\n\n\tStretch line to second point");
    wn_printf(w,"\n\tusing ARROW keys.  Press [ENTER] when done\n");
    while ( (c = getch()) == '\0')          /* if cursor direction key */
    {
        c = getch();                        /* check direction */
        unput_line(x1,y1,x,y);              /* remove old line */
        unput_cursor(x,y);                  /* remove cursor */
        chkkey(c, &x, &y);                  /* get new cursor position */
        put_cursor(x,y);                    /* put cursor back on */
        put_line(x1,y1,x,y);                /* put new line */
        wn_printf(w,"\r\tCoordinates:     X2: %3d   Y2: %3d", x,y);
    } /* end while */
    len = calc_line(x1,y1,x,y);             /* calculate distance */
    wn_printf(w,"\n\n\tFeature is %.1f pixels long.", len);
    if (len > 0.0) {                        /* compare with 5 micron ref */
        VSCALE = len/5.0;                   /* to give Pixel Conversion Factor */
        wn_printf(w,"\n\tFor 5 micron vertical line,");
        wn_printf(w,"\n\tVertical Scale factor is %.4f pixels/micron",VSCALE);
        if (( fp = fopen(session,"a") ) == NULL) {
            wn_printf(w,"\a\n\n\tCANNOT open session file %s",session);
            wn_printf(w,"\n\n\tPress any key to continue");
            getch();
            return(1);                      /* premature termination */
        }
        else {
            fprintf(fp,"\nSETUP:\tMeasured length is %.1f pixels long",len);
            fprintf(fp,"\n\tVertical scale factor is %.4f pixels/micron",VSCALE);
            fclose(fp);                     /* close session file */
        } /* end if-else */
        wn_printf(w,"\n\n\tPress Any key when done ");
        v_kflush();                         /* empty keyboard buffer */
        getch();
    }
    unput_line(x1,y1,x,y);                  /* remove line */
    unput_cursor(x,y);                      /* remove cursor */
    return(0);                              /* everything OK */
}

/*
 * Put out a cross-hair graphic cursor at location x,y, saving the
 * pixels under the graphic cursor.  The intensity of the cursor
 * will be peak-white or peak-black depending on the pixel intensity
 * if the center point.
 */
void
put_cursor(int x,int y)
{
    int  i;                                 /* scratch counter */
    int  x1, y1;                            /* cursor pixel counters */
    int color;                              /* pixel color */

    x1 = x - 4;                             /* get the left position */
    y1 = y - 4;                             /* get the top position */
    /* save pixels under the cursor */
    for ( i = 0; i < 9; i++) {
        xpixval[i] = rpixel(x1+i,y);        /* save pixel value */
        ypixval[i] = rpixel(x,y1+i);
    }
```

```
    if (rpixel(x,y) > GRAY)                 /* enhance visibility */
        color = BLACK_LEVEL;                /* of cursor */
    else
        color = WHITE_LEVEL;
    /* put out the cursor */
    for ( i = 0; i < 9; i++ ) {
          wpixel(x1+i,y,color);             /* draw horizontal */
          wpixel(x,y1+i,color);             /* draw vertical */
    }
}

/*
 * Restore image to original state without cursor
 */
void
unput_cursor(int x,int y)
{
    int i;                                  /* scratch counter */
    int x1,y1;                              /* cursor pixel counters */

    x1 = x - 4;                             /* get left position */
    y1 = y - 4;                             /* get top position */
    for ( i = 0; i < 9; i++ ) {
        wpixel(x1+i,y,xpixval[i]);          /* restore horizontal stroke */
        wpixel(x,y1+i,ypixval[i]);          /* restore vertical stroke */
    }
}

/*
 * check key pressed for direction
 */
void
chkkey(char c, int *x, int *y)
{
    switch (c) {
        case UARROW:                        /* move up */
             (*y)--;
             if ((*y) < IYS) (*y) = LASTROW;/* wrap round */
             break;
        case DARROW:                        /* move down */
             (*y)++;
             if ((*y) > LASTROW) (*y) = IYS;/* wrap round */
             break;
        case LARROW:                        /* move left */
             (*x)--;
             if ((*x) < IXS) (*x) = LASTROW;/* wrap round */
             break;
        case RARROW:                        /* move right */
             (*x)++;
             if ((*x) > LASTCOL) (*x) = IXS;/* wrap round */
             break;
        case PAGEUP:                        /* move up fast */
             (*y) -= STEP;
             if ((*y) < IYS) (*y) = LASTROW;/* wrap round */
             break;
        case PAGEDN:                        /* move down fast */
             (*y) += STEP;
             if ((*y) > LASTROW) (*y) = IYS;/* wrap round */
             break;
        case HOME:                          /* move left fast */
             (*x) -= STEP;
             if ((*x) < IXS) (*x) = LASTROW;/* wrap round */
             break;
        case END:                           /* move right fast */
             (*x) += STEP;
             if ((*x) > LASTCOL) (*x) = IXS;/* wrap round */
             break;
        default:
             break;
```

```
    } /* end switch */
}

/*
 * calculate the distance between two points
 */
float
calc_line(int x1,int y1,int x2,int y2)
{
    if ( abs(x2 - x1) > abs(y2 - y1) )      /* horizontal line */
        return( abs(x2-x1) );
    else                                    /* vertical line */
        return( abs(y2-y1) );
}

/*
 * draw a vertical or horizontal line
 * saving the image pixels underneath
 */
void
put_line(int x1, int y1, int x2, int y2)
{
    int    i;                               /* scratch counter */
    int    color;                           /* line color */

    if (rpixel(x1,y1) > GRAY)               /* enhance visibility */
        color = BLACK_LEVEL;                /* of line */
    else
        color = WHITE_LEVEL;
    if ( abs(x2 - x1) > abs(y2 - y1) ) {    /*  draw horizontal line */
        if (x2 > x1) {
            for (i = x1; i <= x2; i++) {    /* draw left to right */
               line[i-x1] = rpixel(i,y1);   /* save pixel values */
               wpixel(i,y1,color);          /* before overwriting */
            }
        }
        else {                              /* (x2 < x1) */
            for (i = x2; i <= x1; i++) {    /* draw right to left */
               line[i-x2] = rpixel(i,y1);   /* save pixel values */
               wpixel(i,y1,color);          /* before overwriting */
            }
        } /* end if(x2 > x1)-else */
    }
    else {                                  /* draw vertical line */
        if (y2 > y1) {
            for (i = y1; i <= y2; i++) {    /* draw top down */
               line[i-y1] = rpixel(x1,i);   /* save pixel values */
               wpixel(x1,i,color);          /* before overwriting */
            }
        }
        else {                              /* (y2 < y1) */
            for (i = y2; i <= y1; i++) {    /* draw bottom up */
               line[i-y2] = rpixel(x1,i);   /* save pixel values */
               wpixel(x1,i,color);          /* before overwriting */
            }
        } /* end if(Y2 > y1)-else */
    } /* end if(abs(x2 -x1))-else */
}

/*
 * erase a vertical or horizontal line
 * restoring the image pixels underneath
 */
void
unput_line(int x1, int y1, int x2, int y2)
{
    int    i;                               /* scratch counter */

    if ( abs(x2 - x1) > abs(y2 - y1) ) {    /*  erase horizontal line */
```

```
        if (x2 > x1) {
            for (i = x1; i <= x2; i++)          /* erase left to right */
               wpixel(i,y1,line[i-x1]);         /* by restoring original values */
        }
        else {                                  /* (x2 < x1) */
            for (i = x2; i <= x1; i++)          /* erase right to left */
               wpixel(i,y1,line[i-x2]);         /* by restoring original values */
        } /* end if(x2>x1)-else */
    }
    else {                                      /* erase vertical line */
        if (y2 > y1) {
            for (i = y1; i <= y2; i++)          /* erase top down */
               wpixel(x1,i,line[i-y1]);         /* by restoring original values */
        }
        else {                                  /* (y2 < y1) */
            for (i = y2; i <= y1; i++)          /* erase bottom up */
               wpixel(x1,i,line[i-y2]);         /* by restoring original values */
        } /* end if(y2>y1)-else */
    } /* end if(abs(x2-x1))-else */
}

/* End of file SETUP.C */
```

```
/* ************************************************************************
 * FILENAME : ACQUIRE.C
 * CALLED BY: SEMEX main program
 * LAST MODIFIED: 17 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE  : This function allows images to be acquired from the
 *            PCVISIONplus framegrabber board.  After acquiring a frame,
 *            the image can be cropped and inverted if desired before
 *            it is saved.  Values applied to the image are saved in
 *            a session file as well as in the comment line of the image
 *            header.
 * ************************************************************************
 */
#include "global.h"                         /* Necessary defines */
#include <string.h>                         /* string prototypes */

/*
 *  Acquire image from camera, invert, crop and then save it
 */
acquire()
{
    WINDOWPTR wn;                           /* window handle */
    int watrib, batrib;                     /* Window, border and */
    unsigned fatrib,                        /* field attributes */
    char c;                                 /* scratch for user response */
    char s5[5];                             /* scratch string */
    int i;                                  /* scratch index */
    int nocam;                              /* flag indicating camera off */
    int lm, rm;                             /* left and right margin setting */
    int gain, offset;                       /* gain and offset setting */

    watrib = v_setatr(WHITE,BLUE,0,0);      /* set window attribute */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* set border attribute */
    wn = wn_open(500,8,13,50,14,watrib,batrib); /* open dialog window */
    if (!wn) {
       printf("\a\n\n Unable to open window");
       exit(1);
    }
    /*
     * Read in an image or turn on camera
     */
    wn_title(wn," IMAGE ACQUISITION ");
    wn_printf(wn,"\n\n\tRead Image from file [N]?");
    v_kflush();                             /* empty keyboard buffer first */
    c = getch();                            /* get user response */
    if ( (c == 'Y') || (c == 'y') ) {
        getim(wn,0);                        /* read image from file */
        nocam = TRUE;                       /* Note camera is off */
    }
    else {
        wn_puts(wn,2,3,"Turning on GRAB Mode...  ");
        setcamera(0);                       /* connect camera 0 */
        extsync();                          /* external sync */
        grab(NO_WAIT);                      /* ensure grab mode ON */
        grab(NO_WAIT);
        nocam = FALSE;                      /* camera on flag */
        /*
         *  Fine tuning the board's gain and offset.
         */
        wn_title(wn," GAIN AND OFFSET SETTING ");
        gain = GAIN_LVL;
        setgain(gain);                      /* initial gain setting */
        wn_puts(wn,7,1,"Use [+] and [-] keys to adjust");
        wn_puts(wn,8,1,"Press [Enter] to continue");
        if (!DF_GAIN) {
            wn_puts(wn,2,1,"Gain (0 highest, 100 lowest): ");
            wn_printf(wn,"%3d",gain);
            while ( (c=getch()) != ENTER )
            {
```

```
            wn_locate(wn,2,31);
            if ( (c == '+') && (gain < 100) )
              gain = gain + 5;          /* increment in steps of 5 */
            if ( (c == '-') && (gain > 0) )
              gain = gain - 5;          /* decrement in steps of 5 */
            setgain(gain);              /* change the camera gain */
            wn_printf(wn,"%3d",gain);
        }
        GAIN_LVL = gain;                /* update global gain value */
    }
    offset = OFFSET_LVL;
    setoffset(offset);
    if (!DF_OFFSET) {
        wn_puts(wn,4,1,"Offset (0 darkest, 100 lightest): ");
        wn_printf(wn,"%3d",offset);     /* display current offset */
        while ( (c=getch()) != ENTER) {
            wn_locate(wn,4,35);
            if ( (c == '+') && (offset < 100) )
              offset = offset + 5;      /* increment in steps of 5 */
            if ( (c == '-') && (offset > 0) )
              offset = offset - 5;      /* decrement in steps of 5 */
            setoffset(offset);          /* change the camera offset */
            wn_printf(wn,"%3d",offset); /* display new offset */
        }
        OFFSET_LVL = offset;            /* update global offset value */
    }
} /* end if-else */
/*
 *  Snap a frame (this will stop further acquisition).
 */
if (!nocam)
    wn_title(wn," ACQUIRE IMAGE - SNAP MODE ");
while(1) {
    int done = FALSE;                   /* check whether image snapped */
    wn_clr(wn);
    if (!nocam) {                       /* using camera */
        v_kflush();                     /* clear keyboard buffer */
        wn_printf(wn,"\n\n\tPress [SPACEBAR] to snap an image");
        wn_printf(wn,"\n\t  repeat until satisfied");
        while ( (c = getch()) != ENTER) {
            wn_puts(wn,6,4,"Wait...");
            waitvb();                   /* Wait for vertical blanking */
            snap(WAIT);                 /* acquire a frame */
            wn_puts(wn,6,4,"Done!  ");
            wn_puts(wn,12,4,"When Satisfied, press [Enter]");
            done = TRUE;                /* flag that image taken */
        } /* end while */
        if (!done) {
            waitvb();                   /* Wait for vertical blanking */
            snap(WAIT);                 /* turn grab off */
        } /* end if !done */
        crystal();                      /* internal sync for stability */
        setcamera(1);                   /* disconnect camera 0 */
    } /* end if !nocam */
    /* Clear unwanted areas */
    wn_clr(wn);
    wn_printf(wn,"\n\n\tPress [SPACEBAR] to crop image");/* line 2 */
    wn_printf(wn,"\n\t  one vertical line at a time");
    if (DF_LM) {
        if ( LT_MARGIN > IXS ) {        /* auto crop left margin */
            wn_printf(wn,"\n\tCropping left edge...");
            for (i = IXS; i < LT_MARGIN; i++)
                vlclear(i,IYS,NROW,BLACK_LEVEL);
        }
        lm = LT_MARGIN;
    }
    else {                              /* let user crop left margin */
        lm = IXS;                       /* start from left edge */
        wn_puts(wn,5,2,"Left margin: ");                /* line 5 */
```

```
            v_kflush();                     /* clear keyboard buffer */
            wn_printf(wn,"\n\n\tWhen done, press [Enter]"); /* line 7 */
            while ( (c = getch()) != ENTER) {
                vlclear(lm,IYS,NROW,BLACK_LEVEL); /* clear one vertical line */
                wn_locate(wn,5,15);                         /* line 5 */
                wn_printf(wn,"%3d",lm);     /* display current left margin */
                if (lm >= RT_MARGIN)        /* right edge reached */
                    break;                  /* stop */
                else
                    lm++;                   /* do next line */
            } /* end while */
            LT_MARGIN = lm;                 /* update global */
        } /  end if-else */
        if (DF_RM) {
            if(RT_MARGIN < NCOL) {          /* auto crop right margin */
                wn_printf(wn,"\n\tCropping right edge...");
                for (i = NCOL; i > RT_MARGIN; i--)
                    vlclear(i,IYS,NROW,BLACK_LEVEL);
                rm = RT_MARGIN;
            }
        }
        else {                              /* let user crop right margin */
            rm = NCOL;
            v_kflush();                     /* clear keyboard buffer */
            wn_puts(wn,5,2,"Right margin:      ");          /* line 5 */
            wn_printf(wn,"\n\n\tWhen done, press [Enter]"); /* line 7 */
            while ( (c = getch()) != ENTER) {
               vlclear(rm,0,NROW,BLACK_LEVEL); /* clear one vertical line */
               wn_locate(wn,5,16);                          /* line 5 */
               wn_printf(wn,"%3d",rm);      /* display right margin */
               if (rm <= LT_MARGIN)         /* left edge reached */
                 break;                     /* stop */
               else
                 rm--;                      /* do next line */
            } /* end while */
            RT_MARGIN = rm;                 /* update global */
        } /* end if-else */
        if ( (!DF_LM) && (!DF_RM) ) {
            wn_printf(wn,"\n\n\tSatisfied with result [Y]?");/* line 9 */
            v_kflush();                     /* clear keyboard buffer */
            c = getch();                    /* get user response */
            if ( c != 'N' && c != 'n' )     /* not no */
                break;                      /* done, get out of loop */
            /* prepared to redo */
            if (nocam)
                initluts();                 /* restore image */
            else {
                setcamera(0);               /* connect camera 0 */
                extsync();                  /* external sync */
                waitvb();
                grab(NO_WAIT);              /* ensure grab mode ON */
                grab(NO_WAIT);              /* restore image and start over */
            } /* end if-else */
        }
        else
            break;                          /* get out of while loop */
    } /* end while(1) */
    wn_clr(wn);
    /* Complement Image in frame memory */
    if (DF_INVERT) {
        wn_printf(wn,"\n\tComplementing Image...");
        complement(IXS,IYS,NCOL,NROW);
    }
    else {
        wn_printf(wn,"\n\n\tComplement Image [Y]?");
        v_kflush();                     /* clear keyboard buffer */
        c = getch();
        if ( (c!='N') && (c!='n') ) {
            wn_printf(wn,"\n\tComplementing Image...");
```

```
            complement(IXS,IYS,NCOL,NROW);
        }
    } /* end if-else */
    /*
     * Save image
     */
    wn_printf(wn,"\n\n\tUpdating frame memory...");
    maplut(GRNLUT,0,IXS,IYS,NCOL,NROW); /* update frame memory */
    v_kflush();                         /* clear keyboard buffer */
    wn_printf(wn,"\n\n\tSave image[Y] ?");
    c = getch();
    if ( (c != 'n') && (c != 'N') )
        putim(wn,0);                    /* Save image with .img extension */
    /*
     * Update session file
     */
    wn_printf(wn,"\n\n\tUpdating session file...");
    if (( fp = fopen(session,"a") ) == NULL) {
        wn_printf(wn,"\a\n\n\tUnable to open session file %s",session);
        wn_printf(wn,"\n\n\tPress any key to continue");
        getch();
    }
    else {
        if (nocam)
            fprintf(fp,"\nACQUIRE:  Image read from Filename: %s",filename);
        else {
            if ( c != 'n' && c != 'N')
                fprintf(fp,"\nACQUIRE:  Image saved to Filename: %s",filename);
        } /* end if-else */
        fprintf(fp,"\n Comment: %s",comline);
        fclose(fp);                     /* close session file */
    } /* end if-else */
    wn_close(wn);
    return(0);
}

/* end of file ACQUIRE.C */
```

```
/* ************************************************************************
 * FILENAME : CLIP.C
 * CALLED BY: semex main() and calls clipmain()
 * LAST MODIFIED : 12 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE  : This program clips (thresholds) the image on screen by
 *            taking an operator input value and forcing all image pixel
 *            values above the threshold value to BLACK_LEVEL and those
 *            below the threshold value to WHITE_LEVEL.
 *            Latest version has an automatic threshold capability
 *            called autoclip().  It  determines the background value
 *            for predefined 7 regions in the image and uses the
 *            darkest value.
 *            ( Background == WHITE_LEVEL ; Feature == BLACK_LEVEL )
 * ************************************************************************
 */
#include "global.h"

clipmain()
{
    extern int clip(WINDOWPTR w,int t); /* function prototype */
    WINDOWPTR wn;                       /* window pointer */
    int watrib, batrib;                 /* window, border and */
    unsigned fatrib;                    /* field attributes */
    int tval;                           /* threshold value */
    char c;                             /* scratch */

    watrib = v_setatr(WHITE,BLUE,0,0);  /* window color */
    batrib = v_setatr(RED,WHITE,0,BOLD);/* border color */
    wn = wn_open(500,8,13,50,10,watrib,batrib); /* open a dialog window */
    if (!wn) {
        printf("\a\n\tUnable to open window.  Aborting...");
        exit(1);
    }
    wn_title(wn," CLIPPING IMAGE ");
    if ( (fp = fopen(session,"a")) == NULL) {
        wn_printf(wn,"\a\n\n\tCANNOT open session file %s",session);
        wn_printf(wn,"\n\n\tPress any key to continue");
        getch();
        wn_close(wn);                   /* close dialog box */
        return(1);                      /* premature termination */
    }
    if (LOAD_RAW) {                     /* auto loading of RAW image */
        wn_printf(wn,"\n\n\tLoading RAW image");
        if(getim(wn,0))                 /* default ext is .img */
            goto err;                   /* error detected, terminate */
        else
            fprintf(fp,"\nCLIP:\tImage from Filename: %s",filename);
    }
    else {                              /* ask before loading */
        v_kflush();                     /* empty keyboard buffer first */
        wn_printf(wn,"\n\n\tLoad Image from disk [N]? ");
        wn_printf(wn,"\n\tPress [ESC] to QUIT");
        c=getch();
        if (c == ESC) goto err;         /* quit */
        if (c == 'Y'|| c == 'y')
            if(getim(wn,0))             /* default ext is .img */
                goto err;               /* error detected, terminate */
            else
                fprintf(fp,"\nCLIP:\tImage from Filename: %s",filename);
    } /* end if-else */

    /* clip the image */
    wn_clr(wn);
    wn_printf(wn,"\n\n\tDetermining threshold...");
    tval = autoclip();                  /* Use Auto Threshold */
    fprintf(fp,"\n\tAuto Threshold: %3d",tval);
    tval = clip(wn,tval);               /* Modify Threshold */
    fprintf(fp,"\tUser Threshold: %3d", tval);
```

```
    wn_clr(wn);                         /* clear window */
    wn_printf(wn,"\n\n\t1: RESTORE image to original and abort");
    wn_printf(wn,"\n\t2: SAVE the modified image");
    wn_printf(wn,"\n\t3: EXIT without saving");
    wn_printf(wn,"\n\n  Select option by NUMBER [3]: ");
    v_kflush();                         /* empty keyboard buffer first */
    switch (c=getch())
    {
        case '1':
            goto err1;                  /* restore and quit */
        case '2':                       /* update and save */
            wn_printf(wn,"\n\n\tUpdating frame memory");
            maplut(GRNLUT,0,IXS,IYS,NCOL,NROW);
            putim(wn,1);                /* save image with default ext .iml */
            break;
        case '3':                       /* update without saving */
        default:
            wn_printf(wn,"\n\n\tUpdating frame memory");
            maplut(GRNLUT,0,IXS,IYS,NCOL,NROW);
            break;

    } /* end switch */
    linlut(GRNLUT,0);                   /* restore GREEN LUT, bank 0 */
    fclose(fp);                         /* close session file */
    wn_close(wn);                       /* close window */
    return(0);                          /* terminated properly */

    err:                                /* premature termination sequence */
    fprintf(fp,"\n*** CLIP ABORTED ***");
    err1:
    linlut(GRNLUT,0);                   /* restore GREEN LUT, bank 0 */
    fclose(fp);                         /* close session file */
    wn_close(wn);                       /* close window */
    return(1);
}

/*
 * Clip (Threshold) routine creates a binary image.  Pixel values below 'val'
 * are changed to BLACK_LEVEL while all others are changed to WHITE_LEVEL
 * Returns selected threshold.
 * NOTE: The change is not permanent as frame memory is not updated.
 *       The calling program needs to call maplut() to do this.
 */
clip(WINDOWPTR w, int val)              /* passes a window handle */
{
    char c;                             /* scratch */

    wn_puts(w,3,5,"THRESHOLD LEVEL:   ");
    wn_printf(w,"%3d",val);
    wn_puts(w,6,5,"Use [+] and [-] keys to adjust");
    wn_printf(w,"\n\n\tPress [ENTER] when done");
    v_kflush();                         /* empty keyboard buffer first */
    while ((c=getch()) != ENTER)
    {
        wn_locate(w,3,24);
        if ( (c=='+') && (val < HIGH) )
            val += 2;                   /* increase threshold */
        if ( (c=='-') && (val > LOW) )
            val -= 2;                   /* decrease threshold */
        wn_printf(w,"%3d",val);
        threshold(GRNLUT,0,HIGHEST,val);/* threshold GREEN LUT */
    } /* end while */
    return val;                         /* all OK and done */
} /* end clip */

/*
 * Autoclip function - This function samples the background over 7 regions
 *                     of the image and determines the darkest value which
 *                     is returned as the threshold value. The regions are
```

```
 *                    ------------------------------------
 *                    |               (1)                |
 *                    |(2)                           (3)|
 *                    |               (4)                |
 *                    |                                  |
 *                    |(5)            (6)            (7)|
 *                    ------------------------------------
 *
autoclip(void)
{
    int XSl,XEl,XSc,XEc,XSr,XEr;        /* x positions */
    int Ya = 1;                         /* y position for region 1 */
    int Yb = 80;                        /* y position for regions 2 and 3 */
    int Yc = 240;                       /* y position for region 4 */
    int Yd = 478;                       /* y position for regions 5, 6 and 7 */
    int width = 20;                     /* width of regions */
    int CENTER = 245;                   /* center of image */
    int tval;                           /* threshold value */
    int i;                              /* scratch */

    XSl = LT_MARGIN + 1;                /* left edge of regions 1 and 2 */
    XEl = XSl + width;                  /* end of regions 1 and 2 */
    XSc = CENTER;                       /* left edge of central regions */
    XEc = CENTER + width;               /* end of central regions */
    XSr = RT_MARGIN - 1 - width;        /* left edge of regions 6 and 7 */
    XEr = XSr + width;                  /* end of regions 6 and 7 */
    /* determine background intensity for each region */
    tval = WHITE_LEVEL;                 /* initial threshold value */
    tval = findthd(XSc,XEc,Ya,tval);    /* find threshold for region 1 */
    tval = findthd(XSl,XEl,Yb,tval);    /* find threshold for region 2 */
    tval = findthd(XSr,XEr,Yb,tval);    /* find threshold for region 3 */
    tval = findthd(XSc,XEc,Yc,tval);    /* find threshold for region 4 */
    tval = findthd(XSl,XEl,Yd,tval);    /* find threshold for region 5 */
    tval = findthd(XSc,XEc,Yd,tval);    /* find threshold for region 4 */
    tval = findthd(XSr,XEr,Yd,tval);    /* find threshold for region 5 */
    threshold(GRNLUT,0,HIGHEST,tval);   /* threshold GREEN LUT */
    return tval;                        /* return threshold */
}

/*
 * determine background of region defined by the parameters passed
 * returns the darkest background level
 */
findthd(int xs, int xe, int y, int T)
{
    int  pixval;                            /* pixel value */
    int  i;                                 /* scratch counter */

    for (i=xs; i<xe; i++) {
        pixval = rpixel(i,y);               /* read pixel value */
        if( pixval > GRAY )                 /* light pixels assumed to be */
            T = min(T,pixval);              /* background, take darkest */
    }
    return T;                               /* return new threshold */
}

/* end of file CLIP.C */
```

```
/* ************************************************************************
 * FILENAME      : TAG.C
 * DEPENDENCIES  : called by SEMEX main() and calls semio functions
 * LAST MODIFIED : 12 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE: Labels and identifies each feature in the image.  Reads pixel-
 *          by-pixel, left to right, top to bottom, and assigns a unique
 *          ID (fid) number to each feature (agglomeration of pixels) so
 *          that they can be processed by size() or saved for later
 *          processing.  If there are more than 254 features, a group ID
 *          (gid) is also given.  The tag() function requires a binary
 *          image obtained with the clip() function where
 *              Background == WHITE_LEVEL and Feature == BLACK_LEVEL
 ************************************************************************
*/
#include "global.h"                         /* required for all SEMEX files */
#include <time.h>                           /* required for time functions */

static int  fid;                            /* feature indices */
static int  gid;                            /* gid = fid\HIGH */
static int  maxfl;                          /* max feature size */

tagmain()
{
    extern int tag(WINDOWPTR w);            /* prototype declaration */
    WINDOWPTR wn;                           /* window pointer */
    int watrib, batrib;                     /* window, border and */
    char c;

    watrib = v_setatr(WHITE,BLUE,0,0);      /* window color */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* border color */
    wn = wn_open(500,8,13,50,10,watrib,batrib); /* open a dialog window */
    if (!wn) {
        printf("\n\tUnable to open window.  Aborting...");
        exit(1);
    }
    wn_title(wn," TAGGING IMAGE ");
    if ((fp = fopen(session,"a")) == NULL) {
        wn_printf(wn,"\a\n\n\tUnable to open session file %s",session);
        wn_printf(wn,"\n\n\tPress any key to continue");
        getch();
        wn_close(wn);                   /* close dialog box */
        return(1);                      /* premature termination */
    }
    if (LOADCLIP) {                     /* load CLIPPED image without asking */
        wn_printf(wn,"\n\n\tLoading CLIPPED image");
        if(getim(wn,1))                 /* default ext .iml */
            goto err;                   /* error, don't continue */
        else
            fprintf(fp,"\nTAG:\tImage from Filename: %s\n",filename);
    }
    else if (!DO_SEQ) {                 /* not processing frame memory */
        v_kflush();                     /* empty keyboard buffer first */
        wn_printf(wn,"\n\n\tLoad CLIPPED image from disk [N]?");
        wn_printf(wn,"\n\tPress [ESC] to QUIT");
        c=getch();
        if( c == ESC) goto err;         /* quit */
        if( (c == 'Y') || (c == 'y') )
            if(getim(wn,1))             /* default ext .iml */
                goto err;               /* error, don't continue */
            else
                fprintf(fp,"\nTAG:\tImage from Filename: %s\n",filename);
        }
        else fprintf(fp,"\nTAG:");
    /*
     * perform feature extraction and tagging
     */
    if(tag(wn)) {                       /* if error, don't save */
        wn_printf(wn,"\n\n\tPress Any Key to continue");
```

```
        getch();
        goto err;                       /* terminate */
    }
    v_kflush();                         /* empty keyboard buffer first */
    wn_printf(wn,"\n\n\tSave image to Disk File [N]?");
    c=getch();
    if (c== ESC) goto err;              /* quit */
    if( (c == 'Y') || (c == 'y') )
        putim(wn,2);                    /* save with default ext .im2 */
    fclose(fp);                         /* close session file */
    wn_close(wn);                       /* close window */
    return(0);

    err:                                /* error condition */
    fprintf(fp,"\n*** TAG ABORTED ***");
    fclose(fp);
    wn_close(wn);                       /* close window */
    return(1);                          /* signal back error */
}

/*
 * image feature identification and tagging algorithm
 */
tag(WINDOWPTR w)                        /* window handle */
{
    extern void tagrow0(WINDOWPTR w);   /* function prototype */
    extern void tagrows(WINDOWPTR w);   /* declarations */
    extern void checkmerge(WINDOWPTR w);/* ditto */
    extern int tagmerge(WINDOWPTR w);   /* ditto */
    extern void step(void);             /* ditto */
    extern clock_t clock(void);         /* ditto */
    clock_t start;                      /* timing variable */
    float elapsed;                      /* ditto */
    unsigned fatrib;                    /* field attribute */
    char     ibuf[10];                  /* scratch string */
    static char *emsg = "must be between 2 and 255. Press any key";

    fatrib = (BLUE<<4) | WHITE | BOLD;  /* field color */
    wn_clr(w);                          /* clear window */
    if (!DF_SIZE) {
        itoa(OVERSIZE,ibuf,10);         /* convert to string */
        wn_gint(XEQ,NFRM,NFLD,w,2,1,"Eliminate features larger than: ",
                fatrib,'_',&maxfl,3,2,255,ibuf,NSTR,emsg);
    }
    else                                /* use default max */
       maxfl = OVERSIZE;                /* feature size */
    start = clock();                    /* start timer */
    wn_printf(w,"\n\n\tTAGGING FEATURES in Progress\n");
    fid = LOW;                          /* initialize feature count */
    gid = 0;                            /* initialize group count */
    wn_wrap(w,TRUE);                    /* word wrap ON */
    tagrow0(w);                         /* tag top row */
    tagrows(w);                         /* tag subsequent rows */
    fid += gid*HIGH - 1;                /* get total features */
    if (fid < LOW) {
        wn_printf(w,"\a\n\n\tOverflow has occurred.  Fid= %d", fid);
        wn_printf(w,"\n\tAborting Feature Extraction");
        return(1);
    }
    checkmerge(w);                      /* check merging window size */
    if (tagmerge(w))                    /* merge joined features */
        return(1);                      /* signal back error */

    wn_printf(w,"\n\tFEATURE COUNT: %d",fid);
    elapsed = (float) (clock()-start) / (float) CLK_TCK;
    wn_printf(w,"\n\n\tElapsed Time: %.1f seconds",elapsed);
    fprintf(fp,"\tTagged %d features in %.1f seconds",fid,elapsed);
    maplut(GRNLUT,0,IXS,IYS,NCOL,NROW);     /* update frame memory */
    return(0);                          /* signal OK back */
```

```
}

/*
 * Tag the top row of the image
 * Note that fid cannot exceed HIGH. This is taken care of by step()
 */
void
tagrow0(WINDOWPTR w)
{
    extern void step(void);             /* prototype definition */
    register x, rp;                     /* scratch */

    for( x = LT_MARGIN ; x < RT_MARGIN ; x++ )  /* start from left edge */
    {
        if( rpixel(x,IYS) == WHITE_LEVEL )
            continue;                   /* skip background */
        /* from here on, pixel is not background */
        if ( (x != LT_MARGIN) && ((rp=rpixel(x-1,IYS)) != WHITE_LEVEL) ) {
            wpixel(x,IYS,rp);           /* west pixel occupied, adopt ID */
            continue;
        }
        wpixel(x,IYS,fid);              /* no neighbor, give new ID */
        wn_printf(w,"\r\t%5d",fid+gid*HIGH);
        step();                         /* increment feature counter */
    } /* end for x */
}

/*
 * Tag subsequent rows in the image
 * Note that fid cannot exceed HIGH. This is taken care of by step()
 */
void
tagrows(WINDOWPTR w)
{
    extern void step(void);
    register x1;                        /* scratch */
    register x, y;                      /* current pixel location */
    int rp;                             /* pixel value */

    for( y = 1 ; y < NROW ; y++ )       /* start from top edge + 1 */
    {
        for( x = LT_MARGIN ; x < RT_MARGIN ; x++ ) /* start from left edge */
        {
            if( rpixel(x,y) == WHITE_LEVEL )
                continue;               /* skip background */
            /* from here on, pixel is a feature */
            if( (x != LT_MARGIN) && ((rp=rpixel(x-1,y)) != WHITE_LEVEL) ) {
                wpixel(x,y,rp);         /* west pixel occupied, adopt ID */
                continue;
            }
            if( (rp=rpixel(x,y-1)) != WHITE_LEVEL ) {
                wpixel(x,y,rp);         /* north pixel occupied, adopt ID */
                continue;
            }
            wpixel(x,y,fid);            /* no N/W neighbors, give new ID */
            /* Check rest of row for connectivity */
            x1 = x + 1;                 /* start with east neighbor */
            while(1)
            {
                if( rpixel(x1,y) == WHITE_LEVEL ) { /* end found */
                    wpixel(x,y,fid);    /* assign ID */
                    wn_printf(w,"\r\t%5d",fid+gid*HIGH);
                    step();             /* increment feature counters */
                    break;              /* done */
                }
                if( (rp=rpixel(x1,y-1)) != WHITE_LEVEL ) { /* merge */
                    wpixel(x,y,rp);     /* use north pixel's ID */
                    break;
                }
```

```
                xl++;                       /* move further east */
            } /* end while */
        }  /* end for x */
    } /* end for y */
}

/*
 * Checks reliability of merge algorithm by comparing the maximum feature
 * length, maxfl, and the density of the features
 */
void
checkmerge(WINDOWPTR w)
{
    int safesize;                           /* safe max window size */

    safesize = (int) (NROW /( (int)(fid/(HIGH+1)) + 2));
    if (maxfl > safesize) {
        wn_clr(w);                          /* declutter the screen */
        wn_printf(w,"\a\n\tSizing window cannot support %d features",fid);
        wn_printf(w,"\n\tShrinking window from %d to %d pixels",maxfl,safesize);
        wn_printf(w,"\n\tAny larger features will be removed");
        wn_printf(w,"\n\tIf this happens, Clip the image again");
        wn_printf(w,"\n\tat a lower threshold so as to reduce");
        wn_printf(w,"\n\tthe number of features.");
        wn_printf(w,"\n\n\tPress any key to continue");
        maxfl = safesize;
        OVERSIZE = safesize;
        getch();
    }
    else wn_printf(w,"\n\tLargest permissible feature is %d pixels",safesize);
}

/*
 * The following algorithm to merge joined features will fail if
 * maxfl is set too large and there are many (>254) small features
 * bunched together.
 */
tagmerge(WINDOWPTR w)
{

   register  xl,yl;                         /* in-window variables */
   int     x,y;                             /* current pixel indices */
   int     id;                              /* current pixel ID */
   int     nid;                             /* ID of pixel above */
   int     xleft,xright,ytop,ybot;          /* sizing window limits */
   int     found, merge_done, nmerged = 0; /* tracks merging */

   wn_printf(w,"\n\tCombining joined features...\n");
   for( y = 1 ; y < NROW ; y++ )            /* start from row 1 */
   {
       ytop = y - maxfl;                    /* set row search limits */
       ybot = y + maxfl;
       if( ytop < 1) ytop = 1;
       if( ybot > NROW ) ybot = NROW;
       for( x = LT_MARGIN ; x < RT_MARGIN ; x++ )  /* left edge to right */
       {
          id = rpixel(x,y);                 /* get current pixel ID */
          nid = rpixel(x,y-1);              /* get ID of pixel above */
          if( (id == WHITE_LEVEL) || (nid == WHITE_LEVEL) ||
              (id == nid) )                 /* skip if background or */
              continue;                     /* part of same feature */
          /* Joined features exists.  Merge */
          found = FALSE;                    /* exist but not found */
          xleft = x - maxfl;                /* set column search limits */
          xright = x + maxfl;
          if( xleft < LT_MARGIN) xleft = LT_MARGIN;
          if( xright > RT_MARGIN ) xright = RT_MARGIN;
          for ( yl = ytop ; yl < ybot ; yl++ ) {   /* scan vertically */
             merge_done = TRUE,                    /* assume merged */
```

```
            for ( x1 = xleft, x1 < xright; x1++)  /* scan horizontally */
              if ( rpixel(x1,y1) == id ) {        /* same feature */
                 found = TRUE;                    /* found the culprit */
                 wpixel(x1,y1,nid);               /* change its ID */
                 merge_done = FALSE;              /* assumption wrong */
              }
            if (found && merge_done) break; /* culprit found and merged */
          } /* end for y1 */
          /* one feature has been merged */
          wn_printf(w,"\r\t%5d", ++nmerged);
          fid--;                                /* decrement feature count */
          if (fid < LOW) {                      /* check */
              wn_printf(w,"\a\n\n\tUnderflow has occurred. Fid = %d", fid);
              wn_printf(w,"\n\tAborting Feature Extraction");
              return(1);                        /* signal error back */
          }
        } /* end for x */
    } /* end for y */
    TOTAL = (long)fid;                          /* save the grand total */
    return(0);                                  /* signal OK back */
}

/*
 * step() allocates a unique pair of numbers to each feature, namely
 * a feature ID number (fid) and a group ID number (gid).  Allocation
 * is based on the limitation that fid must not exceed 8 bits.  Therefore
 * Increment fid until it reaches HIGH
 * then reset it to LOW and increment gid
 * This is necessary as pixel value is 8 bits only
 */
void
step(void)
{
    fid++;
    if( fid > HIGH ) {
        fid = LOW;
        gid++;
    }
}


/* End of file TAG.C */
```

```
/* ************************************************************************
 * FILENAME : SIZE.C
 * CALLED BY: semex main() and calls semio.c functions
 * LAST MODIFIED : 12 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE  :  This routine uses an existing or saved image that has
 *             been clipped with clip() and tagged with tag().  The output
 *             of this program is a tabular output of the calculated
 *             area, X-Chord, and Y-Chord that is suitable for
 *             input to a statistical analysis program.
 * NOTES:   :  TOTAL (no of features) is automatically set by TAG.
 *             Background == WHITE_LEVEL
 *             Feature == BLACK_LEVEL
 * ************************************************************************
 */
#include "global.h"                         /* required by all SEMEX files */
#include <time.h>                           /* required for time functions */
#include <string.h>                         /* required for string functions */

static float  yscale;                       /* vertical scale factor */
static int  fid;                            /* track number of features */
static int  maxfl, minfl;                   /* tracks max and min features */
static int  xmax, ymax;                     /* max feature lengths */
static int  xmin, ymin;                     /* min feature lengths */
static long minarea, maxarea;               /* smallest and largest areas */
static long *aptr;                          /* pointer to area store */
static int  *xptr, *yptr;                   /* pointer to x,y stores */

sizemain()
{
    extern int size(WINDOWPTR w);           /* prototype declaration */
    WINDOWPTR wn;                           /* window pointer */
    int watrib, batrib;                     /* window, border and */
    unsigned fatrib;                        /* field attributes */
    char c;

    watrib = v_setatr(WHITE_BLUE,0,0);      /* window color */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* border color */
    wn = wn_open(500,8,12,50,10,watrib,batrib); /* open a dialog window */
    if (!wn)
    {
        printf("\n\tUnable to open window.  Aborting...");
        exit(1);
    }
    wn_title(wn," SIZING FEATURES ");
    if ((fp = fopen(session,"a")) == NULL) {
        wn_printf(wn,"\a\n\n\tUnable to open session file %s",session);
        wn_printf(wn,"\n\n\tPress any key to continue");
        getch();
        wn_close(wn);                       /* close dialog box */
        return(1);                          /* premature termination */
    }
    if (LOADTAG) {                          /* load TAGGED image w/o asking */
        wn_printf(wn,"\n\n\tLoading TAGGED image");
        if (getim(wn,2))                    /* if error don't size */
            goto err;                       /* terminate */
        else
            fprintf(fp,"\nSIZE \tImage from Filename: %s\n",filename);
    }
    else if (!DO_SEQ) {                     /* not processing frame memory */
        wn_printf(wn,"\n\n\tLoad TAGGED image from disk [N]?");
        wn_printf(wn,"\n\tPress [ESC] to QUIT");
        v_kflush();                         /* empty keyboard buffer first */
        c = getch();
        if( c == ESC) goto err;             /* quit */
        if(c == 'Y'|| c == 'y')             /* read image with ext .im2 */
            if (getim(wn,2))                /* if error don't size */
                goto err;                   /* terminate */
            else
```

```
            fprintf(fp,"\nSIZE:\tImage from Filename: %s\n",filename);
        }
        else fprintf(fp,"\nSIZE:");          /* process frame memory contents */
    /* feature sizing */
    if(size(wn)) {                          /* if error don't save */
        wn_printf(wn,"\n\n\tPress any Key to continue");
        getch();
        goto err;                           /* terminate */
    }
    wn_printf(wn,"\n\n\tSave data [Y]?");
    v_kflush();                             /* empty keyboard buffer first */
    c = getch();                            /* get user response */
    if ( (c!='N') && (c!='n') ) outdata();  /* display data */
    fclose(fp);                             /* close session file */
    /* free dynamically allocated memory */
    free(xptr);
    free(yptr);
    free(aptr);
    v_kflush();                             /* empty keyboard buffer first */
    wn_printf(wn,"\n\n\tSave image to Disk File [N]?");
    c=getch();                              /* get user response */
    if(c == 'Y'|| c == 'y') {
        wn_printf(wn,"\n\n\tUpdating frame memory...");
        maplut(GRNLUT,0,IXS,IYS,NCOL,NROW); /* update frame memory */
        putim(wn,3);                        /* save image with ext .im3 */
    }
    wn_close(wn);                           /* done, close window */
    return(0);

    err:                                    /* terminate phase */
    fprintf(fp,"\n*** SIZE ABORTED ***");
    fclose(fp);                             /* close session file */
    wn_close(wn);                           /* close window */
    return(1);
}

/*  feature sizing algorithm  */
size(WINDOWPTR w)                           /* pass window pointer */
{
    extern clock_t clock(void);             /* function prototypes */
    extern int outdata(void);               /* ditto */
    extern void pixelsize(WINDOWPTR w);     /* ditto */
    clock_t start;                          /* timing variable */
    float elapsed;                          /* ditto */
    unsigned fatrib;                        /* field attributes */
    char c;                                 /* scratch */
    char ubuff[20];                         /* scratch string buffers */
    /* help message strings */
    static char *hlp1 = "Sets vertical pixel scale factor [1]";
    static char *hlp2 = "Features larger than this will be discarded [100]";
    static char *hlp3 = "Features smaller than this will be discarded [1]";
    /* error message strings */
    static char *emsg1 = "must be between 0 and 100. Press any key";
    static char *emsg2 = "must be between 2 and 256. Press any key";
    static char *emsg3 = "must be between 1 and 255. Press any key";
    static char *emsg4 = "must be between 100 and 9999. Press any key";

    wn_clr(w);                              /* clear window */
    fatrib = (BLUE<<4) | WHITE | BOLD;      /* field color */
    if (DF_VSCALE)
        yscale = VSCALE;                    /* use default conversion */
    else {
        sprintf(ubuff,"%7.2f",VSCALE);      /* convert to string */
        wn_gfloat(XEQ,NFRM,NFLD,w,2,2,"Vertical Scale factor: ",
              fatrib,'_',&yscale,7,2,0.0,100.0,ubuff,hlp1,emsg1);
    }
    if (DF_SIZE) {                          /* set feature limits */
        minf1 = UNDERSIZE;
        maxf1 = OVERSIZE;
```

```
    }
    else {
        itoa(UNDERSIZE,ubuff,10);           /* convert to string */
        wn_gint(XEQ,NFRM,NFLD,w,4,2,  "Discard Features SMALLER than: ",
            fatrib,'_',&minfl,3,1,255,ubuff,hlp3,emsg3);
        itoa(OVERSIZE,ubuff,10);            /* convert to string */
        wn_gint(XEQ,NFRM,NFLD,w,6,2,  "Discard Features GREATER than: ",
            fatrib,'_',&maxfl,3,2,256,ubuff,hlp2,emsg2);
    }
    /* Dynamically allocate memory for the TOTAL number of features
     * generated by TAG.  If this is not available, user option is allowed.
     */
    if (TOTAL <= 0) {
        wn_printf(w,"\n\n\tImage not recently tagged.  Continue [N] ?");
        v_kflush();                         /* prevent spurious input */
        c = getch();                        /* get user response */
        if ((c =='Y') || (c == 'y')) {
            itoa(2000,ubuff,10);
            wn_gint(XEQ,NFRM,NFLD,w,8,2,  "Max Number of Features Expected: ",
                fatrib,'_',&TOTAL,4,100,9999,ubuff,hlp2,emsg4);
        }
        else return(1);                     /* don't size */
    } /* end if */
    xptr = (int *) calloc(TOTAL,sizeof(int));
    yptr = (int *) calloc(TOTAL,sizeof(int));
    aptr = (long *) calloc(TOTAL,sizeof(long));
    /* Check for successfull memory allocation   */
    if( !xptr || !yptr || !aptr ) {
        wn_printf(w,"\a\n\tNot enough Memory to allocate!");
        wn_printf(w,"\n\tReduce TOTAL = %d and try again", TOTAL);
        wn_printf(w,"\n\n\tPress any key to continue");
        getch();
        return(1);
    }
    fprintf(fp,"\tVertical scale: %f pixels/unit length",yscale);
    fprintf(fp,"\n\tMin Length spec: %4d\tMax Length spec: %4d",minfl,maxfl);
    /* Begin Sizing Routine   */
    start = clock();                        /* start timing */
    wn_wrap(w,TRUE);                        /* turn wordwrap ON */
    wn_printf(w,"\n\n\tSizing features\n");
    pixelsize(w);                           /* size image in pixels */
    elapsed = (float) (clock()-start) / (float) CLK_TCK;
    wn_printf(w,"\n\n\tElapsed time: %.1f seconds", elapsed );
    fprintf(fp,"\n\tSizing took %.1f seconds",elapsed);
    return(0);                              /* signal OK back */
}

/*
 * Size features by pixel count. No scaling factors taken into account.
 */
void
pixelsize(WINDOWPTR w)
{
    register int  x1, y1;                   /* scratch indices */
    int  x, y;                              /* current pixel location */
    int  currval;                           /* current pixel value */
    int  xleft, ytop, xright, ybot;         /* sizing box */
    int  xlen=0, ylen=0;                    /* zeros feature dimensions */
    long pixarea=0;                         /* zeros feature areas */
    int  reject=0;                          /* zeros rejects */
    int  too_large=0;                       /* zeros oversized features */
    int  too_small=0;                       /* zeros undersized features */

    fid = 0;                                /* Initialize Feature counter  */
    xmax = 0;                               /* Max X-Chord reset to zero */
    ymax = 0;                               /* Max Y-Chord reset to zero */
    xmin = NCOL;                            /* Min X-Chord reset to max */
    ymin = NROW;                            /* Min Y-Chord reset to max */
    minarea = NCOL * NROW;                  /* Min area reset to max */
```

```
maxarea = 0;                            /* Max area reset to zero */
for( y = IYS ; y < NROW ; y++ )         /* do for all rows */
{
    if( fid == TOTAL ) break;           /* Quit when all features sized */
    ytop = y;                           /* set up y coordinates */
    ybot = y + maxfl;                   /* for sizing box */
    if( ytop < IYS) ytop = IYS;
    if( ybot > NROW ) ybot = NROW;
    for( x = LT_MARGIN; x < RT_MARGIN; x++ )  /* do for all columns */
    {
        currval = rpixel(x,y);          /* get current pixel value */
        if( fid == TOTAL )break;        /* all features sized */
        if( (currval == WHITE_LEVEL) || (currval == BLACK_LEVEL) )
            continue;

        /* pixel is part of feature yet to be sized */
        xleft = x - maxfl;              /* Set up x coordinates */
        xright = x + maxfl;             /* for sizing box */
        if( xleft < LT_MARGIN) xleft = LT_MARGIN;
        if( xright > RT_MARGIN ) xright = RT_MARGIN;
        for( y1 = ytop ; y1 < ybot ; y1++ )  /* do for all rows in box */
        {
            for( x1 = xleft ; x1 < xright ; x1++ ) /* do for columns */
            {
                if( rpixel(x1,y1) != currval ) continue; /* skip */
                pixarea++;              /* increment pixel area */
                xlen++;                 /* increment x length */
                wpixel(x1,y1,BLACK_LEVEL);  /* mark off as counted */
            } /* end for x1 */
            if( xlen==0 ) break;        /* passed bottom edge */
            if( xlen >  xptr[fid] )     /* update max x length */
                xptr[fid] = xlen;
            xlen = 0;                   /* reset x length */
            ylen++;                     /* increment y length */
        } /* end for y1 */
        yptr[fid] = ylen;               /* store max y length */
        aptr[fid] = pixarea;            /* store max area */
        ylen = 0;                       /* ready for next feature */
        pixarea = 0;
        /* Collect statistics for rejects */
        if ( xptr[fid] < minfl || yptr[fid] < minfl ) {
            too_small++;
            reject++;
            xptr[fid] = 0;              /* re-initialize x element */
            yptr[fid] = 0;              /* re-initialize y element */
            continue;                   /* don't increment fid */
        }
        if( xptr[fid] >= maxfl || yptr[fid] >= maxfl) {
            too_large++;
            reject++;
            xptr[fid] = 0;              /* re-initialize x element */
            yptr[fid] = 0;              /* re-initialize y element */
            continue;                   /* don't increment fid */
        }
        /* Calculate Min/Max values  */
        xmax = max( xmax, xptr[fid] );  /* widest feature */
        ymax = max( ymax, yptr[fid] );  /* tallest feature */
        xmin = min( xmin, xptr[fid] ) ; /* narrowest feature */
        ymin = min( ymin, yptr[fid] ) ; /* shortest feature */
        maxarea = max( maxarea, aptr[fid] ) ; /* largest feature */
        minarea = min( minarea, aptr[fid] ) ; /* smallest feature */
        fid++;                          /* get next feature */
        wn_printf(w,"\r\t%5d",fid);
    } /* end for x */
} /* end for y */
TOTAL = (long)fid;                      /* Record new total */
wn_clr(w);                              /* clear dialog box */
wn_printf(w,"\n\n\tSized %ld Features within specifications",TOTAL);
fprintf(fp,"\n\t%ld Features were within specifications",TOTAL);
```

```
    if (too_small > 0) {
        wn_printf(w,"\n\t%d Features were less than %d pixels",too_small,minfl);
        fprintf(fp,"\n\t%d Features were less than %d pixels",too_small,minfl);
    }
    if (too_large > 0) {
        wn_printf(w,"\n\t%d Features were greater than %d pixels",too_large,maxfl);
        fprintf(fp,"\n\t%d Features were greater than %d pixels",too_large,maxfl);
    }
    if (reject > 0) {
        wn_printf(w,"\n\t%d Features were REJECTED",reject);
        fprintf(fp,"\n\t%d Features were REJECTED",reject);
    }
}

/*
 * Output routine to display x and y dimensions, and 2 areas
 * AREA1 assumes feature is elliptical and uses PI*xlen*ylen
 * AREA2 converts directly from pixel area to feature area.
 */
outdata(void)
{
    const float PI4 = 0.785398;             /* define pi/4 */
    FILE  *fdata;                           /* data file pointer */
    WINDOWPTR w;                            /* window pointer */
    int watrib, batrib;                     /* window, border and */
    unsigned fatrib;                        /* field attributes */
    int  j;                                 /* scratch index */
    float  Cx, Cy, Ca;                      /* Conversion constants */
    float  fxmax, fymax, fxmin, fymin;      /* scaled statistics */
    float  xlen, ylen;                      /* scaled dimensions */
    float  area, fmaxarea, fminarea;        /* scaled areas */
    char ubuff[20], datafile[20];           /* scratch string buffers */
    char c;

    watrib = v_setatr(WHITE,BLUE,0,0);      /* window color */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* border color */
    w = wn_open(800,0,13,50,23,watrib,batrib); /* open a dialog window */
    if (!w)
    {
        printf("\n\tUnable to open window.  Aborting...");
        exit(1);
    }
    wn_title(w," TABLE OF FEATURE DATA ");
    /* Calculate Conversion Constants required to properly scale and convert
     * pixels to dimensioned units.  Depends on VSCALE and ASPECT_RATIO.
     * ASPECT_RATIO is defined in SEMEX.C
     */
    Cx = ASPECT_RATIO/yscale;               /* x conversion constant */
    Cy = 1.0/yscale;                        /* y conversion constant */
    Ca = Cx*Cy;                             /* area conversion constant */
    fprintf(fp,"\n\tConversion constants: Cx=%f  Cy=%f  Ca=%f", Cx, Cy, Ca);
    wn_printf(w,"\n\n\tImage filename: %s",filename);
    chgext(datafile,filename,".dat");       /* change extension to .dat */
    fatrib = (BLUE<<4) | WHITE | BOLD;      /* field color */
    wn_gtext(XEQ,NFRM,NFLD,w,4,1,"Save DATA in filename: ",
             fatrib,'_',18,datafile,NSTR,NSTR);
    wn_printf(w,"\n\n\tSending output to FILE %s\n",datafile);
    if( ( fdata = fopen(datafile,"w") ) == NULL )  {
        wn_printf(w,"\a\n\n\tCANNOT Open output file %s\n", datafile);
        wn_printf(w,"\n\tPress Any Key to continue");
        getch();
        wn_close(w);
        return(1);
    }
    for( j = 0 ; j < TOTAL ; j++ ) {
        xlen = xptr[j] * Cx;                /* true x length */
        ylen = yptr[j] * Cy;                /* true y length */
        fprintf(fdata,"%6d %10.3f %10.3f  %8.3f %8.3f\n", j+1, PI4*xlen*ylen,*(aptr+j)*Ca,xlen,ylen);
    }
```

```
    wn_printf(w,"\n ID NO     AREA_C     AREA_M  X-Chord  Y-Chord\n");
    for( j = 0 ; j < TOTAL ; j++ ) {
        xlen = xptr[j] * Cx;                /* true x length */
        ylen = yptr[j] * Cy;                /* true y length */
        wn_printf(w,"\n%6d %10.3f %10.3f %8.3f %8.3f", j+1, PI4*xlen*ylen, *(aptr+j)*Ca, xlen, ylen );
        if ( (j%20) == 19 ) {               /* pause on full page */
            wn_printf(w,"\n\tPress [ENTER] for MORE or [ESC] to QUIT");
            if ((c = getch()) == ESC) break;/* quit if ESC pressed */
            if (j != TOTAL - 1)             /* not end yet */
                wn_printf(w,'\n ID NO     AREA_C     AREA_M  X-Chord  Y-Chord\n");
        }
    }
    fclose(fdata);                          /* close data file */
    /* display statistics */
    xlen = xmax*Cx;
    ylen = ymax*Cy;
    area = maxarea*Ca;
    wn_printf(w,"\n\n          AREA_M      X-Chord      Y-Chord");
    wn_printf(w,  "\n  Max   %10.3f   %10.3f   %10.3f", area, xlen, ylen);
    fprintf(fp,   "\n\t          AREA_M      X-Chord      Y-Chord");
    fprintf(fp,   "\n\tMax   %10.3f   %10.3f   %10.3f", area, xlen, ylen);
    xlen = xmin*Cx;
    ylen = ymin*Cy;
    area = minarea*Ca;
    wn_printf(w,  "\n  Min   %10.3f   %10.3f   %10.3f", area, xlen, ylen);
    fprintf(fp,   "\n\tMin   %10.3f   %10.3f   %10.3f\n", area, xlen, ylen);
    wn_printf(w,"\n\n\tPress Any Key to continue");
    v_kflush();
    getch();
    wn_close(w);
    return(0);                              /* signal healthy end */
}

/* End of file SIZE.C */
```

```
/* ************************************************************************
 * FILENAME: ANALYZE.C
 * CALLED BY: SEMEX main()
 * LAST MODIFIED: 14 Mar 91 by LEE, YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE: This set of routines analyzes the data files put out by
 *          the size() functions.  It first allows the user to specify
 *          the data files and then merges the data from them into an
 *          array.  The volume and data are then calculated and a
 *          histogram built using histo_vol().  This can be plotted
 *          using SEM.M inside MATLAB.
 * ************************************************************************
 */

#include "global.h"                     /* required by all SEMEX modules */
#include <dos.h>                        /* dos prototype definitions */
#include <math.h>                       /* math prototype definitions */
#include <string.h>                     /* string handling prototype defn */

typedef struct {                        /* structured list of file pointers */
    char name[13];                      /* string to contain data file */
} SLIST;

typedef struct {                        /* Matlab MAT-file structure */
    long type;                          /* type */
    long mrows;                         /* row dimension */
    long ncols;                         /* column dimension */
    long imagf;                         /* flag indicating imag part */
    long namlen;                        /* name length (including NULL) */
} Fmatrix;

static SLIST *list;                     /* Define pointer to list */
static char sdate[13];                  /* date string */
static int fatrib;                      /* field attribute */
static float *cptr;                     /* pointer to area store */
static size_t memsize;                  /* size of memory allocated */
static int Nfiles;                      /* number of data files */

analyze()
{
    extern int merge_data(WINDOWPTR wn);    /* prototype definition */
    extern void histo_vol(WINDOWPTR wn);    /* ditto */
    WINDOWPTR wn;                           /* window handle */
    int watrib, batrib;                     /* window and border attributes */
    char c;                                 /* scratch for user response */

    watrib = v_setatr(WHITE,BLUE,0,0);      /* window attribute */
    batrib = v_setatr(RED,WHITE,0,BOLD);    /* border attribute */
    fatrib = (BLUE << 4 ) | WHITE | BOLD;   /* field attribute defined */
    wn = wn_open(500,8,13,52,13,watrib,batrib); /* open dialog window */
    if(!wn) {
        printf("\a\nUnable to open window. Aborting...");
        exit(1);
    }
    wn_title(wn," ANALYSE FEATURES ");
    if ( (fp = fopen(session,"a")) == NULL) {
        wn_printf(wn,"\a\n\n\tUnable to open session file %s",session);
        wn_printf(wn,"\n\n\tPress any key");
        getch();
        wn_close(wn);
        return(1);
    }
    else                                    /* session file opened */
        fprintf(fp,"\nANALYZE:Merging data files");
    if (merge_data(wn))                     /* merge data files */
        goto err;                           /* terminate if error */
    histo_vol(wn);                          /* histogram the data */
    fclose(fp);                             /* close session file */
    wn_close(wn);                           /* close window */
```

```
    return(0);

    err:                                    /* error trapped */
    fprintf(fp,"\n*** ANALYZE ABORTED ***");
    fclose(fp);                             /* close session file */
    wn_close(wn);                           /* close window */
    return(0);
}

/* merge_data() prompts the user with all files with extension .dat from
 * which to merged data.  Selected files are written into a list
 */
merge_data(WINDOWPTR wn)
{
    extern void get_dates(unsigned date);   /* prototype definition */
    extern int extract_data(WINDOWPTR wn, int Ncol); /* ditto */
    char c;                                 /* scratch for user response */
    int Area_type;                          /* defines type of area to use */
    struct find_t d_file;                   /* structure of data files */

    wn_title(wn," SELECT DATA FILES ");
    Nfiles = 0;                             /* initialize file counter */
    wn_clr(wn);                             /* clear dialog box */
    _dos_findfirst("*.dat",_A_NORMAL, &d_file);  /* get first occurrance */
    get_dates(d_file.wr_date);              /* returns date string to sdate */
    wn_printf(wn,"\n\n\t DATA FILENAME  DATE CREATED");
    wn_locate(wn,4,4);                      /* place cursor at row,col */
    wn_printf(wn," %12s  %s  \n\n\tInclude[Y] ?",d_file.name,sdate);
    v_kflush();                             /* empty keyboard buffer */
    c = getch();                            /* get user response */
    if ( (c=='Y') || (c=='y') || (c==ENTER) ) {
        if ((list = (SLIST *)calloc(1,sizeof(SLIST))) == NULL) { /* allocate memory */
            wn_printf(wn,"\a\n\n\tNo memory to allocate");
            wn_printf(wn,"\n\n\tPress any key to continue");
            getch();
            return(1);                      /* signal error back */
        }
        strcpy(list[Nfiles].name,d_file.name); /* save filename into list */
        Nfiles++;                           /* increment file count */
    }
    /* find the rest of the data files */
    while (_dos_findnext(&d_file) == 0) {
        get_dates(d_file.wr_date);          /* returns date string to sdate */
        wn_locate(wn,4,4);                  /* place cursor at row,col */
        wn_printf(wn," %12s  %s  \n\n\tFile Count: %3d\tInclude[Y] ?",d_file.name,sdate,Nfiles);
        v_kflush();                         /* empty keyboard buffer */
        c = getch();                        /* get user response */
        if ( (c=='Y') || (c=='y') || (c==ENTER)) {
            if ((list = (SLIST *)realloc((void *)list,(Nfiles+1)*sizeof(SLIST))) == NULL) {
                wn_printf(wn,"\a\n\n\tNo memory to reallocate");
                wn_printf(wn,"\n\n\tPress any key");
                getch();
                goto err1;                  /* abort */
            }
            strcpy(list[Nfiles].name,d_file.name); /* save filename into list */
            Nfiles++;                       /* increment file counter */
        }
        if ( c == ESC ) break;              /* enough already */
    } /* end while */
    wn_printf(wn,"\n\n %d datafiles selected in this directory",Nfiles);
    wn_printf(wn,"\n Press [Enter] to Histogram data or [Esc] to Abort");
    v_kflush();                             /* flush keyboard buffer */
    if ( getch() == ESC)                    /* get user response */
        goto err1;                          /* abort */
    /* allocate memory for first element */
    memsize = sizeof(float);
    if ((cptr = (float *)calloc(1,memsize)) == NULL) {
        wn_printf(wn,"\a\n\n\tNo memory to allocate");
        wn_printf(wn,"\n\n\tPress any key");
```

```
        getch();
        goto err1;                          /* signal error back */
    }
    wn_printf(wn,"\n\n\tUse C)alculated or M)easured area [C] ?");
    v_kflush();                             /* clear keyboard buffer */
    c = getch();                            /* get user response */
    if ( (c == 'M') || (c == 'm') ) {
        Area_type = 2;                      /* use measured area AREA_M */
        fprintf(fp,"\n\tExtracting MEASURED Area from");
    }
    else {
        Area_type = 1;                      /* use calculated area AREA_C */
        fprintf(fp,"\n\tExtracting CALCULATED Area from");
    }
    extract_data(wn,Area_type);             /* extract area from col 2 */
    return(0);                              /* signal back OK */

    err1:                                   /* termination sequence */
    free(list);                             /* deallocate memory */
    return(1);                              /* signal back abort */
}

/*
 * Get date string given a dos date code
 * returns date string to sdate declared static
 *
 * date format is:  Bits 0 - 4 : Day of the month (value between 1 and 31)
 *                       5 - 8 : Month (value between 1 and 12)
 *                       9 - 15: Year since 1980
 */
void
get_dates(unsigned date)
{
    char *mth;                              /* month string */
    unsigned mm,dd,yy;                      /* date variables */
    static char *month[12] = { "Jan","Feb","Mar","Apr","May","Jun",
                               "Jul","Aug","Sep","Oct","Nov","Dec" };

    dd = date & 0x001f;                     /* get bits 0 to 4 */
    mm = (date & 0x01e0) >> 5;              /* get bits 5 to 8 */
    yy = (date & 0xfe00) >> 9;              /* get bits 9 to 15 */
    sprintf(sdate,"%2u %3s %4u",dd,month[mm-1],yy+1980);
}
```

```
/* extract_data() extracts a single column of data from a specified file
 */
extract_data(WINDOWPTR wn, int Ncol)
{
    FILE *dfp;                              /* data file pointer */
    float col[5];                           /* scratch array */
    int fid;                                /* scratch feature counter */
    int i;                                  /* scratch counter */

    wn_clr(wn);
    wn_title(wn,"   EXTRACTING DATA   ");
    TOTAL = 0;                              /* initialize total no of features */
    for (i=0; i < Nfiles; i++) {            /* read a data file */
        if ((dfp = fopen(list[i].name,"r")) == NULL) {  /* read data within */
            wn_printf(wn,"\a\n\n\tCANNOT Open datafile %s",list[i].name);
            wn_printf(wn,"\n\n\tPress any key to continue");
            getch();
            return(1);                      /* error recovery */
        }
        wn_printf(wn,"\n\t%s", list[i].name); /* display data file */
        fprintf(fp,"\n\t\t%s", list[i].name);   /* record date file name */
        while (fscanf(dfp," %d %f %f %f %f ", &fid,&col[1],&col[2],&col[3],&col[4]) != EOF)
        {   /* read in data */
            cptr[TOTAL] = col[Ncol];        /* use requested column */
            TOTAL++;                        /* increment total # */
            /* allocate memory for 1 more element */
            memsize += sizeof(float);       /* increase memory counter */
            if (( cptr = (float *) realloc((void *) cptr, memsize )) == NULL) {
                wn_printf(wn,"\a\n\n\tReallocation Failed.  Out of memory.");
                wn_printf(wn,"\n\n\tPress any key");
                getch();
                fclose(dfp);                /* close data file */
                return(1);                  /* error recovery */
            }
        } /* end while fscan dfp */
        fclose(dfp);
        wn_printf(wn,"\n%10d extracted.  Running Total is %ld", fid, TOTAL);
        fprintf(fp,"%10d extracted.  Running Total is %ld", fid, TOTAL);
    } /* end for loop */
    v_kflush();                             /* empty keyboard buffer */
    wn_printf(wn,"\n\n\tPress [Enter] to Histogram or [Esc] to Quit");
    free(list);                             /* deallocate memory */
    if ( getch() == ESC ) return(1);        /* quit if ESC pressed */
    return(0);                              /* successful */
}

/* histo_vol() uses the merged data from merge_data(), converts it to volume
 * and collates the data into 38 bins so that a histogram is obtained.
 */
#define BINS  38                            /* No of bins */

void
histo_vol(WINDOWPTR wn)
{
    extern void savemat(FILE *fptr, int type, char *pname, /* prototype definition */
                        int mrows, int ncols, int imagf, /* for saving data */
                        double *preal, double *pimag);   /* in MATLAB form */
    char histfile[20];                      /* histogram file */
    FILE *fh;                               /* file pointer */
    double PI = 3.141593;                   /* define pi */
    double UPPER = 180.0;                   /* upper limit of histogram */
    double STEP = 0.8264;                   /* same as Malvern MasterSizer */
    static double volume[BINS];             /* bins to contain volumes */
    static long count[BINS];                /* bins to contain counts */
    double limit[BINS+1];                   /* array of limit values */
    double tot_vol = 0.0;                   /* total volume */
    double toosmall = 0.0;                  /* bin to contain small particles */
    int reject = 0;                         /* track rejects */
    double diam, vol;                       /* diameter and volume variables */
```

```
double C;                               /* constant = 4/3pi */
double R2;                              /* scratch = Area/PI */
int i;                                  /* scratch index counters */
long j;                                 /* scratch particle counter */
double Nf, T;                           /* scratch for type conversion */
char c;                                 /* scratch for user response */

wn_clr(wn);
wn_title(wn," HISTOGRAM VOLUMES ");
C = PI * 4.0 / 3.0;                     /* constant of proportionality */
limit[0] = UPPER;                       /* assign upper limit to element 0 */
for (i=0; i < BINS; i++) {
    limit[i+1] = limit[i] * STEP;       /* generate limits for each bin */
    volume[i] = 0.0;                    /* initialize volume bin */
    count[i] = 0;                       /* initialize count bin */
}
wn_printf(wn,"\n\n\tCalculating equivalent volumes...");
for (j=0; j < TOTAL; j++) {             /* repeat for all particles */
    /* calculate the equivalent volume assuming a sphere, given area */
    R2 = cptr[j]/PI;                    /* square of the radius */
    diam = 2.0 * sqrt(R2);              /* diameter of particle */
    vol  = C * pow(R2,1.5);             /* volume of particle */
    tot_vol += vol;                     /* accumulate total volume */
    /* sieve the volumes and collate into the correct bins */
    if (diam > UPPER) {                 /* too large */
        wn_printf(wn,"\n\tMassive particle Volume=%g, Size=%f",vol,diam);
        fprintf(fp,"\n\tMassive particle Volume=%g, Size=%f",vol,diam);
        continue;
    }
    i = 0;
    while (TRUE) {                      /* repeat until right bin found */
        if (diam > limit[i+1]) {        /* found */
            volume[i] += vol;           /* accumulate volume in bin */
            count[i]++;                 /* increment count */
            break;                      /* get out of while loop */
        }
        else {
            i++;                        /* check next smaller bin */
            if (i >= BINS) {            /* no more bins */
                toosmall += vol;        /* put these into toosmall */
                reject++;               /* increment reject */
                break;                  /* get out of while loop */
            }
        } /* end if-else */
    } /* end while */
} /* end for j */
fprintf(fp,"\n\t%d particles < %f rejected.  Volume= %g or %6.3f%%",reject,limit[BINS],toosmall,
        toosmall*100/tot_vol);
/* display results */
wn_printf(wn,"\n BIN  UPPER  LOWER     Volume  %% Total Vol  Count");
for (i=0; i < BINS; i++) {
    if (limit[i] > 100.0)               /* use one decimal point only */
        wn_printf(wn,"\n %3d  %5.1f  %5.1f %10.2f      %6.2f   %5ld",
            i+1,limit[i],limit[i+1],volume[i],volume[i]*100.0/tot_vol,count[i]);
    else                                /* use two decimal points */
        wn_printf(wn,"\n %3d  %5.2f  %5.2f %10.2f      %6.2f   %5ld",
            i+1,limit[i],limit[i+1],volume[i],volume[i]*100.0/tot_vol,count[i]);
    if (i%10 == 9) {                    /* 10 bins at a time */
        wn_printf(wn,"\n\n\tPress [Enter] for More");
        v_kflush();                     /* empty keyboard buffer */
        getch();                        /* wait for user response */
        if (i != BINS - 1) {            /* not last bin yet */
            wn_clr(wn);                 /* generate new screenful */
            wn_printf(wn,"\n BIN  UPPER  LOWER     Volume  %% Total Vol  Count");
        }
    }
}
/* Print out results */
chgext(histfile,session,".his");        /* form histogram file name */
```

```
    wn_printf(wn,"\n\n\tPrinting results to %s",histfile);
    fprintf(fp,"\n\tPrinting results to %s",histfile);
    if ((fh = fopen(histfile,"a")) == NULL) {
        wn_printf(wn,"\a\n\n\tCANNOT Open file %s",histfile);
        wn_printf(wn,"\n\n\tPress any key");
        getch();
        goto err3;                          /* premature termination */
    }
    fprintf(fh,"\n %s:  Data from %d data files.  Total Particle count: %ld",histfile,Nfiles,TOTAL);
    fprintf(fh,"\n BIN  UPPER  LOWER      Volume  %% Total Vol  Count  %% Total");
    for (i=0; i < BINS; i++)                /* write to histogram file */
        if (limit[i] > 100.0)               /* use one decimal point only */
            fprintf(fh,"\n %3d   %5.1f   %5.1f %10.2f        %6.2f    %5ld    %6.2f",i+1, limit[i],
                limit[i+1],volume[i],volume[i]*100.0/tot_vol,count[i],count[i]*100.0/TOTAL);
        else                                /* use two decimal points */
            fprintf(fh,"\n %3d  %5.2f  %5.2f  %10.2f         %6.2f     %5ld     %6.2f", i+1,limit[i],
                limit[i+1],volume[i],volume[i]*100.0/tot_vol,count[i],count[i]*100.0/TOTAL);
    fclose(fh);                             /* close histfile */
    wn_printf(wn,"\n\t%d Files extracted; Particle Count = %ld",Nfiles,TOTAL);
    wn_printf(wn,"\n\tGenerate a MATLAB MAT-file [Y] ?");
    v_kflush();                             /* prevent spurious inputs */
    c = getch();                            /* get user response */
    if ( (c != 'N') && (c !='n')) {         /* generate MAT-file */
        chgext(histfile,session,".mat");
        wn_clr(wn);
        wn_gtext(XEQ,NFRM,NFLD,wn,4,4,"MATLAB MAT-filename: ",fatrib, '_', 18, histfile,NSTR,NSTR);
        wn_printf(wn,"\n\n\tPrinting MAT-file %s",histfile);
        fprintf(fp,"\n\tPrinting MAT-file %s",histfile);
        if ((fh = fopen(histfile,"w+b")) == NULL) {
            wn_printf(wn,"\a\n\n\tCANNOT Open file %s",histfile);
            wn_printf(wn,"\n\n\tPress any key");
            getch();
            goto err3;                      /* premature termination */
        }
        Nf = (double)Nfiles;                /* for proper type casting */
        T  = (double)TOTAL;                 /* ditto */
        /*  MATLAB variables will be:
         *      'Nfiles','total','limit', 'totalv' and 'volume'
         */
        savemat(fh, 0, "Nfiles", 1, 1, 0, &Nf, (double *)0.0);
        savemat(fh, 0, "total", 1, 1, 0, &T, (double *)0.0);
        savemat(fh, 0, "limit", 1, BINS+1, 0, limit, (double *)0.0);
        savemat(fh, 0, "volume", 1, BINS, 0, volume, (double *)0.0);
        savemat(fh, 0, "totalv", 1, 1, 0, &tot_vol, (double *)0.0);
        savemat(fh, 0, "count", 1, BINS, 0, (double *)count, (double *)0.0);
        fclose(fh);                         /* close MAT-file */
    }
    err3:                                   /* terminate stage */
    free(cptr);                             /* deallocate dynamic memory */
}
```

```
/*
 * savemat - C language routine to save a matrix in a MAT-file.
 * Author J.N. Little 11-3-86
 * Adapted by Y.L. Lee 25 Feb 91
 *
void
savemat(fptr, type, pname, mrows, ncols, imagf, preal, pimag)
FILE *fptr;                                  /* File pointer */
int type;                                    /* Type flag: 0 for PC */
                                             /* Add 1 for text variables. */
                                             /* See LOAD for more info. */
int mrows;                                   /* row dimension */
int ncols;                                   /* column dimension */
int imagf;                                   /* imaginary flag */
char *pname;                                 /* pointer to matrix name */
double *preal;                               /* pointer to real data */
double *pimag;                               /* pointer to imag data */
{
      Fmatrix x;
      int mn;

    x.type = (long) type;
    x.mrows = (long) mrows;
    x.ncols = (long) ncols;
    x.imagf = (long) imagf;
    x.namlen = (long) (strlen(pname) + 1);
      mn = x.mrows * x.ncols;

    fwrite(&x, sizeof(Fmatrix), 1, fptr);
    fwrite(pname, sizeof(char), (int)x.namlen, fptr);
    fwrite(preal, sizeof(double), mn, fptr);
      if (imagf) {
        fwrite(pimag, sizeof(double), mn, fptr);
      }
}


/* End of file ANALYZE.C */
```

```
/* ************************************************************************
 * FILENAME : SEMIO.C
 * CALLED BY: Various SEMEX functions
 * LAST MODIFIED : 7 Mar 91 by LEE YEAW-LIP
 * ------------------------------------------------------------------------
 * PURPOSE  : getim(w,n) - reads in an image from disk
 *            putim(w,n) - saves an image to disk
 *            where n = 0 appends .img to filename (raw image)
 *                      1 appends .im1 to filename (clipped)
 *                      2 appends .im2 to filename (tagged)
 *                      3 appends .im3 to filename (sized)
 *            chgext(*fd,*fs,*ext) - replaces the extension in the source
 *                                   and copies it to the destination
 *
 * NOTE:  All these functions have been declared globally
 ************************************************************************
 */
#include "global.h"                         /* required by all SEMEX files */
#include <string.h>                         /* string handling prototypes */
#include <math.h>                           /* math function prototypes */

static char *digit = "1234567890";          /* define a digit */
/*
 * Read an image from disk
 * passes a window handle and the default file extension where
 *      0: .img,  1: .im1,  2: .im2,  3: .im3
 */
getim(WINDOWPTR w,int n)
{
    int   errval;                           /* scratch for errors */
    char  c, *line2;                        /* scratch */
    char  *margins;                         /* loc of margins in comline */
    char  *vs;                              /* loc of VSCALE in comline */
    char  *lm, *rm;                         /* left and right margin loc */
    static char fbuf[MAXFLEN];              /* filename buffer */
    unsigned fatrib;                        /* field attribute */

    wn_clr(w);                              /* clear screen */
    wn_printf(w,"\n\tREAD IMAGE FROM FILE ");
    strcpy(fbuf,filename);                  /* make copy of filename */
    while(1)
    {
        switch (n)                          /* decide which default ext */
        {                                   /* to use */
            case 1:
                chgext(filename,fbuf,".im1");   /* clipped image */
                break;
            case 2:
                chgext(filename,fbuf,".im2");   /* tagged image */
                break;
            case 3:
                chgext(filename,fbuf,".im3");   /* sized image */
                break;
            case 0:
            default:
                chgext(filename,fbuf,".img");   /* raw video image */
                break;
        } /* end switch */
        fatrib = (BLUE<<4) | WHITE | BOLD;  /* field color */
        wn_gtext(XEQ,NFRM,NFLD,w,2,1,"Filename: ",fatrib,
                 '_',18,filename,NSTR,NSTR);
        errval = readim(IXS,IYS,NCOL,NROW,filename,comline);
        if(errval == 0) {                   /* image successfully read */
            wn_wrap(w,TRUE);                /* allow for word wrap */
            wn_printf(w,"\n COMMENTS: \n %s",comline);
            /* check for 2nd comment line */
            if ((line2 = strrchr(comline,'\n')) != NULL) {
                /* Locate the start of the keyword "Margins=" */
                if ((margins = strstr(line2,"Margins=")) != NULL)
```

```
            {   /* margins exists, extract them */
                margins = strpbrk(margins,digit);   /* locate starting digit */
                LT_MARGIN = atoi(margins);          /* extract left margin */
                margins = strpbrk(margins,",");     /* locate separator */
                /* skip over comma and extract right margin */
                RT_MARGIN = atoi(++margins);        /* extract right margin */
            }
             * locate the start of the keywork "VSCALE" */
            if ((vs = strstr(line2,"VSCALE=")) != NULL)
            {   /* VSCALE exists, extract it */
                vs = strpbrk(vs,digit);         /* locate starting digit */
                VSCALE = (float) atof(vs);      /* vertical scale factor */
            }
        }
        wn_printf(w,"\n\n\tPress any key to continue");
        v_kflush();                         /* prevent premature keystroke */
        getch();
        return(0);                          /* all is well */
    }
    else {                                  /* problem */
        switch(errval)
        {
            case FILE_ERROR:
                wn_printf(w,"\n\tError opening file\n");
                break;
            case FORMAT_ERROR:
                wn_printf(w,"\n\tUnknown file format\n");
                break;
            case READ_ERROR:
                wn_printf(w,"\n\tError Reading file\n");
                break;
            default:
                wn_printf(w,"\n\tUnknown Error %d\n",errval);
                break;
        } /* end switch */
        wn_printf(w,"\a\n\tTry Again [Y]?\n");
        v_kflush();                         /* empty keyboard buffer first */
        c = getch();                        /* get response */
        if(c == 'N' || c == 'n') return(1); /* signal problem back */
    } /* end if-else */
    wn_clr(w);                              /* clear window and try again */
  } /* end while */
}

/*
 * Save an image to disk
 * passes a window handle and file extension where
 *      0: .img,  1: .im1,  2: .im2,  3: .im3
 */
putim(WINDOWPTR w,int n)
{
    unsigned fatrib;                        /* field attribute */
    int  errval;                            /* scratch for error handling */
    char c;                                 /* scratch */
    static char fbuf[MAXFLEN], ext[5];      /* scratch strings */

    wn_clr(w);                              /* clear window */
    wn_printf(w,"\n\tSAVE IMAGE TO DISK ");
    strcpy(fbuf,filename);                  /* make copy of filename */
    while(1)
    {
        switch (n)                          /* decide which ext to use */
        {
            case 1:
                strcpy(ext,".im1");         /* append default ext .im1 */
                chgext(filename,fbuf,ext);
                break;
            case 2:
                strcpy(ext,".im2");         /* append default ext .im2 */
```

```
                chgext(filename,fbuf,ext);
                break;
            case 3:
                strcpy(ext,".im3");          /* append default ext .im3 */
                chgext(filename,fbuf,ext);
                break;
            case 0:
            default:
                strcpy(ext,".img");          /* append default ext .im4 */
                chgext(filename,fbuf,ext);
                break;
        } /* end switch */
        fatrib = (BLUE<<4) | WHITE | BOLD;  /* field color */
        wn_gtext(XEQ,NFRM,NFLD,w,2,1,"FILENAME: ",
                 fatrib,'_',18,filename,NSTR,NSTR);
        /* append image defaults into second comment line */
        sprintf(comline2," Gain= %3d; Offset= %3d; Margins= %3d, %3d;        VSCALE= %7.3f",
                         GAIN_LVL,OFFSET_LVL,LT_MARGIN,RT_MARGIN,VSCALE);
        wn_wrap(w,TRUE);                     /* allow for word wrap */
        wn_printf(w,"\n COMMENTS:\n\n%s",comline2);
        wn_gtext(XEQ,NFRM,NFLD,w,4,1,NSTR,fatrib,'_',48,comline1,NSTR,NSTR);
        strcpy(comline,comline1);            /* copy first line of comment */
        strcat(comline,"\n");
        strcat(comline,comline2);            /* append additional comments */
        wn_printf(w,"\n\n\n\n\tStoring Image...");
        errval = saveim(IXS,IYS,NCOL,NROW,COMPRESSION,filename,comline);
        if(errval == 0) {
            wn_printf(w,"\n\n\tImage successfully SAVED");
            return(0);                       /* all is well */
        }
        else {
            wn_printf(w,"\a\n\tError saving file!!");
            if(errval == ALLOCATION_ERROR)
                wn_printf(w,"\n\tInsufficient Disk Space");
            if(errval == WRITE_ERROR)
                wn_printf(w,"\n\tError writing file or values");
            wn_printf(w,"\n\n\tTry Again [Y]?");
            v_kflush();                      /* empty keyboard buffer first */
            c = getch();
            if(c == 'N' || c == 'n') return(1); /* signal problem back */
            wn_clr(w);
        } /* end if-else */
        wn_clr(w);                           /* clear window and try again */
    } /* end while */
} /* end putim */

/* Takes the source filename, fs, strips it of its extension, copies it to
 * the destination filename, fd, and appends the new extension, ext, to the end.
 */
void
chgext(char *fd,char *fs,char *ext)
{
    char *period;               /* indicates position of the period */
    int len;                    /* gives length of filename less extension */

    period = strrchr(fs,'.');
    len = strlen(fs) - strlen(period);
    strncpy(fd,fs,len);
    fd[len] = '\0';
    strcat(fd,ext);
}

/* End of file SEMIO.C */
```

```
% MATLAB Script file SEM.M used for plotting
%   Histogram of Particle Volume
% Input variables are
%       total     : Total Number of Particles
%       totalv    : Total Volume of Particles
%       Nfiles    : Number of data files merged
%       limit     : array containing the limits of the bins
%       volume    : array containing the volume in each bin

clear
clg
clc

BINS = 38;      % No of bins
fprintf('\n\nPLOT HISTOGRAM FUNCTION\n\n');
file = input('MAT-filename to plot: ','s');
eval(['load ' file]);
zs = 0;
ans = input('Do you want to input Malvern data [Y/N]? ','s');
if (ans == 'Y' | ans == 'y')
        format compact
        ms = zeros(1:BINS);     % allocate BINS
        for i = 1:31            % Malvern has 31 bins only
                ms(i) = input([num2str(i) '. Range ' num2str(limit(i)) ' - ' ...
                               num2str(limit(i+1)) ' Percent Vol = ']);
        end
        for i = BINS:-1:1       % form histogram for malvern data
                zi = ms(i);
                zs = [zs zi zi 0];
        end
end
xs = limit(BINS+1);
ys = 0;
for i = BINS:-1:1               % form histogram for SEMEX data
   xh = limit(i);
   xl = limit(i+1);
   yi = volume(i)*100/totalv;
   xs = [xs xl xh xh];
   ys = [ys yi yi 0];
end
ymax = max([ys zs]);
ypos = ymax/20;
axis([-1, 2.3, 0, max(ymax)+ypos]);
if (ans == 'Y' | ans == 'y')
        semilogx(xs,ys,'-',xs,zs,'--');
        text(30,ymax-ypos*7,'--- Malvern');
        text(30,ymax-ypos*8,'__ SEMEX');
else
        semilogx(xs,ys);
end
title('HISTOGRAM OF PARTICLE VOLUME');
xlabel('Particle Size in Microns (Log scale)');
ylabel('Percent of Total Volume');
text(30,ymax-ypos*2, ['Merged from ' num2str(Nfiles) ' images'])
text(30,ymax-ypos*3, ['Filename: ' file '.mat'])
text(30,ymax-ypos*4, ['Total Vol: ' num2str(totalv) ' um3'])
text(30,ymax-ypos*5, ['Particle Count: ' num2str(total)])
axis;

# End of file SEM.M
```

LIST OF REFERENCES

1. D. W. Netzer, and J. P. Powers, "Experimental Techniques for Obtaining Particle Behavior in Solid Propellant Combustion," paper presented at AGARD, *66th Specialists' Meeting on Smokeless Propellants,* Florence, Italy, September 1985.

2. D. W. Humphries, "Mensuration Methods in Optical Microscopy," *Advances in Optical and Electron Microscopy*, v. 5, Barer R. and Cosslett V. E., eds., pp. 42-69, Academic Press, New York, 1973.

3. D. N. Redman, *Image Analysis of Solid Propellant Combustion Holograms using an ImageAction Software Package,* Master's Thesis, Naval Postgraduate School, Monterey, California, January 1986.

4. E. S. Orguc, *Automatic Data Retrieval From Rocket Motor Holograms,* Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

5. D. S. Kaeser, *Code Optimization of Speckle Reduction Algorithms for Image Processing of Rocket Motor Holograms,* Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

6. V. R. Hockgraver, *Implementation of ImageActionplus Software for Image Analysis of Solid Propellant Combustion Holograms,* Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1989.

7. J. P. Powers, *Automatic Particle Sizing From Rocket Motor Holograms,* Technical Report NPS EC-91-003, Naval Postgraduate School, Monterey, CA, December 1990.

8. Malvern Instruments, *Malvern Particle Sizer Reference Manual,* Version 3.0, 1986.

9. A. Gany, and D. W. Netzer, "Combustion Studies of Metallized Fuels for Solid-Fuel Ramjets," *Journal of Propulsion and Power,* v. 2, No. 5, pp. 423-427, September-October 1986.

10. E. D. Youngborg, T. E. Pruitt, M. J. Smith, and D. W. Netzer, "Light Diffraction Particle Size Measurements in Small Solid Propellant Rockets," *Journal of Propulsion and Power,* December 1989.

11. L. J. Kellman, *An Experimental Validation of a Combined Optical and Collection Probe for Solid Propellant Exhaust Particle Analysis,* Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1991.

12. Technical Publications Dept., *PCVISIONplus Frame Grabber User's Manual,* Image Technology Inc., Woburn, Massachusetts, April 1987.

13. P. Mongelluzzo, *The Window BOSS,* Star Guidance Consulting, Inc., Waterbury, Connecticut, August 1988.

14. Technical Publications Dept., *ITEX PCplus Programmer's Manual,* Imaging Technology Inc., Woburn, Massachusetts, April 1987.

15. Gahm, J., "Instruments for Stereometric Analysis with the Microscope - Their Application and Accuracy of Measurement," *Advances in Optical and Electron Microscopy,* v. 5, Barer, R., and Cosslett, V. E., eds., pp. 115-161, Academic Press, New York, 1969.

DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	1
4.	Professor John P. Powers, Code EC/Po Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5002	4
5.	Professor David W. Netzer, Code AA/Nt Naval Postgraduate School Monterey, California 93943-5002	2
6.	Dr. Michael Holmes, AL/LSNE Air Force Phillips Laboratory Edwards AFB, California 93523-5000	2
7.	Mr. Lee Yeaw-Lip Air Logistics Department, HQ RSAF Mindef Building, Gombak Drive, Republic of Singapore, SE 2366	2